MySQL Reference Manual

Version: 3.23.3-alpha 16 September 1999

1 General Information about MySQL

This is the **MySQL** reference manual; it documents **MySQL** version 3.23.3-alpha.

MySQL is a very fast, multi-threaded, multi-user and robust SQL (Structured Query Language) database server.

For Unix and OS/2 platforms, **MySQL** is basically free; for Microsoft platforms you must get a **MySQL** license after a trial time of 30 days. See Chapter 3 [Licensing and Support], page 20.

The **MySQL** home page provides the latest information about **MySQL**.

For a discussion of **MySQL**'s capabilities, see Section 1.4 [Features], page 4.

For installation instructions, see Chapter 4 [Installing], page 29. For tips on porting **MySQL** to new machines or operating systems, see Appendix G [Porting], page 443.

For information about upgrading from a 3.21 release, see Section 4.16.2 [Upgrading-from-3.21], page 83.

For a tutorial introduction to MySQL, see Chapter 8 [Tutorial], page 208.

For examples of SQL and benchmarking information, see the benchmarking directory. For source distributions, this is the '`bench`' directory. For binary distributions, this is the '`sql-bench`' directory.

For a history of new features and bug fixes, see Appendix D [News], page 400.

For a list of currently known bugs and misfeatures, see Appendix E [Bugs], page 438.

For future plans, see Appendix F [TODO], page 439.

For a list of all the contributors to this product, see Appendix C [Credits], page 394.

IMPORTANT:

Send bug (error) reports, questions and comments to the mailing list at `mysql@lists.mysql.com`. See Section 2.3 [Bug reports], page 15.

For source distributions, the `mysqlbug` script can be found in the '`scripts`' directory. For binary distributions, `mysqlbug` can be found in the '`bin`' directory.

If you have any suggestions concerning additions or corrections to this manual, please send them to the **MySQL** mailing list (`mysql@lists.mysql.com`) with the following subject line: `documentation suggestion: [Insert Topic Here]`. See Section 2.1 [Mailing-list], page 14.

1.1 What is MySQL?

MySQL is a true multi-user, multi-threaded SQL database server. SQL is the most popular database language in the world. **MySQL** is a client/server implementation that consists of a server daemon `mysqld` and many different client programs and libraries.

SQL is a standardized language that makes it easy to store, update and access information. For example, you can use SQL to retrieve product information and store customer information for a web site. **MySQL** is also fast and flexible enough to allow you to store logs and pictures in it.

The main goals of **MySQL** are speed, robustness and ease of use. **MySQL** was originally developed because we at TcX needed a SQL server that could handle very large databases an order of magnitude faster than what any database vendor could offer to us. We have now been using **MySQL** since 1996 in an environment with more than 40 databases containing 10,000 tables, of which more than 500 have more than 7 million rows. This is about 100 gigabytes of mission-critical data.

The base upon which **MySQL** is built is a set of routines that have been used in a highly demanding production environment for many years. Although **MySQL** is still under development, it already offers a rich and highly useful function set.

The official way to pronounce **MySQL** is "My Ess Que Ell" (Not MY-SEQUEL).

1.2 About this manual

This manual is currently available in Texinfo, plain text, Info, HTML, PostScript and PDF versions. Because of their size, PostScript and PDF versions are not included with the main **MySQL** distribution, but are available for separate download at `http://www.mysql.com`.

The primary document is the Texinfo file. The HTML version is produced automatically with a modified version of `texi2html`. The plain text and Info versions are produced with `makeinfo`. The Postscript version is produced using `texi2dvi` and `dvips`. The PDF version is produced with the Ghostscript utility `ps2pdf`.

This manual is written and maintained by David Axmark, Michael (Monty) Widenius, Paul DuBois and Kim Aldale. For other contributors, see Appendix C [Credits], page 394.

1.2.1 Conventions used in this manual

This manual uses certain typographical conventions:

`constant` Constant-width font is used for command names and options; SQL statements; database, table and column names; C and Perl code; and environment variables. Example: "To see how `mysqladmin` works, invoke it with the `--help` option."

'`filename`' Constant-width font with surrounding quotes is used for filenames and pathnames. Example: "The distribution is installed under the '`/usr/local/`' directory."

'`c`' Constant-width font with surrounding quotes is also used to indicate character sequences. Example: "To specify a wildcard, use the '`%`' character."

italic Italic font is used for emphasis, *like this*.

boldface Boldface font is used for access privilege names (e.g., "do not grant the **process** privilege lightly") and to convey **especially strong emphasis**.

When commands are shown that are meant to be executed by a particular program, the program is indicated by the prompt shown with the command. For example, `shell>` indicates a command that you execute from your login shell, and `mysql>` indicates a command that you execute from the `mysql` client:

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

Shell commands are shown using Bourne shell syntax. If you are using a `csh`-style shell, you may need to issue commands slightly differently. For example, the sequence to set an environment variable and run a command looks like this in Bourne shell syntax:

```
shell> VARNAME=value some_command
```

For `csh`, you would execute the sequence like this:

```
shell> setenv VARNAME value
shell> some_command
```

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

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

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

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

SQL statements may be written in uppercase or lowercase. When this manual shows a SQL statement, uppercase is used for particular keywords if those keywords are under discussion (to emphasize them) and lowercase is used for the rest of the statement. So you might see the following in a discussion of the `SELECT` statement:

```
mysql> SELECT count(*) FROM tbl_name;
```

On the other hand, in a discussion of the `COUNT()` function, the statement would be written like this:

```
mysql> select COUNT(*) from tbl_name;
```

If no particular emphasis is intended, all keywords are written uniformly in uppercase.

In syntax descriptions, square brackets (‘`[`’ and ‘`]`’) are used to indicate optional words or clauses:

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

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

```
TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)
{DESCRIBE | DESC} tbl_name {col_name | wild}
```

1.3 History of MySQL

We once started off with the intention of using `mSQL` to connect to our tables using our own fast low-level (ISAM) routines. However, after some testing we came to the conclusion that `mSQL` was not fast enough or flexible enough for our needs. This resulted in a new SQL interface to our database but with almost the same API interface as `mSQL`. This API was chosen to ease porting of third-party code.

The derivation of the name **MySQL** is not perfectly clear. Our base directory and a large number of our libraries and tools have had the prefix "my" for well over 10 years. However, Monty's daughter (some years younger) is also named My. So which of the two gave its name to **MySQL** is still a mystery, even for us.

1.4 The main features of MySQL

The following list describes some of the important characteristics of **MySQL**:

- Fully multi-threaded using kernel threads. That means it easily can use multiple CPUs if available.
- C, C++, Eiffel, Java, Perl, PHP, Python and TCL APIs. See Chapter 20 [Clients], page 324.
- Works on many different platforms. See Section 4.2 [Which OS], page 32.
- Many column types: signed/unsigned integers 1, 2, 3, 4 and 8 bytes long, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `TEXT`, `BLOB`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`, `SET` and `ENUM` types. See Section 7.2 [Column types], page 119.
- Very fast joins using an optimized one-sweep multi-join.
- Full operator and function support in the `SELECT` and `WHERE` parts of queries. Example:

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

- SQL functions are implemented through a highly-optimized class library and should be as fast as they can get! Usually there shouldn't be any memory allocation at all after query initialization.
- Full support for SQL `GROUP BY` and `ORDER BY` clauses. Support for group functions (`COUNT()`, `COUNT(DISTINCT)`, `AVG()`, `STD()`, `SUM()`, `MAX()` and `MIN()`).
- Support for `LEFT OUTER JOIN` with ANSI SQL and ODBC syntax.
- You can mix tables from different databases in the same query (as of version 3.22).
- A privilege and password system which is very flexible and secure, and which allows host-based verification. Passwords are secure since all password traffic when connecting to a server is encrypted.
- ODBC (Open-DataBase-Connectivity) for Windows95 (with source). All ODBC 2.5 functions and many others. You can, for example, use Access to connect to your **MySQL** server. See Chapter 16 [ODBC], page 302.
- Very fast B-tree disk tables with index compression.
- 16 indexes per table are allowed. Each index may consist of 1 to 15 columns or parts of columns. The maximum index length is 256 bytes (this may be changed when compiling **MySQL**). An index may use a prefix of a `CHAR` or `VARCHAR` field.
- Fixed-length and variable-length records.
- In-memory hash tables which are used as temporary tables.
- Handles large databases. We are using **MySQL** with some databases that contain 50,000,000 records.

- All columns have default values. You can use `INSERT` to insert a subset of a table's columns; those columns that are not explicitly given values are set to their default values.
- Uses GNU Automake, Autoconf, and `libtool` for portability.
- Written in C and C++. Tested with a broad range of different compilers.
- A very fast thread-based memory allocation system.
- No memory leaks. Tested with a commercial memory leakage detector (`purify`).
- Includes `isamchk`, a very fast utility for table checking, optimization and repair. See Chapter 13 [Maintenance], page 280.
- Full support for the ISO-8859-1 Latin1 character set. For example, the Scandinavian characters å, ä and ö are allowed in table and column names.
- All data are saved in ISO-8859-1 Latin1 format. All comparisons for normal string columns are case insensitive.
- Sorting is done according to the ISO-8859-1 Latin1 character set (the Swedish way at the moment). It is possible to change this in the source by adding new sort order arrays. To see an example of very advanced sorting, look at the Czech sorting code. **MySQL** supports many different character sets that can be specified at compile time.
- Aliases on tables and columns as in the SQL92 standard.
- `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` return how many rows were changed (affected).
- Function names do not clash with table or column names. For example, `ABS` is a valid column name. The only restriction is that for a function call, no spaces are allowed between the function name and the '`(`' that follows it. See Section 7.30 [Reserved words], page 206.
- All **MySQL** programs can be invoked with the `--help` or `-?` options to obtain online assistance.
- The server can provide error messages to clients in many languages. See Section 9.1 [Languages], page 240.
- Clients connect to the **MySQL** server using TCP/IP connections or Unix sockets, or named pipes under NT.
- The **MySQL**-specific `SHOW` command can be used to retrieve information about databases, tables and indexes. The `EXPLAIN` command can be used to determine how the optimizer resolves a query.

1.5 How stable is MySQL?

This section addresses the questions, "how stable is **MySQL**?" and, "can I depend on **MySQL** in this project?" Here we will try to clarify some issues and to answer some of the more important questions that seem to concern many people. This section has been put together from information gathered from the mailing list (which is very active in reporting bugs).

At TcX, **MySQL** has worked without any problems in our projects since mid-1996. When **MySQL** was released to a wider public, we noticed that there were some pieces of "untested code" that were quickly found by the new users who made queries in a manner different

than our own. Each new release has had fewer portability problems than the previous one (even though each has had many new features), and we hope that it will be possible to label one of the next releases "stable".

Each release of **MySQL** has been usable and there have been problems only when users start to use code from "the gray zones". Naturally, outside users can't know what the gray zones are; this section attempts to indicate those that are currently known. The descriptions deal with the 3.22.x version of **MySQL**. All known and reported bugs are fixed in the latest version, with the exception of the bugs listed in the bugs section, which are things that are "design"-related. See Appendix E [Bugs], page 438.

MySQL is written in multiple layers and different independent modules. These modules are listed below with an indication of how well-tested each of them is:

The ISAM table handler — Stable
: This manages storage and retrieval of all data in **MySQL** 3.22 and earlier versions. In all **MySQL** releases there hasn't been a single (reported) bug in this code. The only known way to get a corrupted table is to kill the server in the middle of an update. Even that is unlikely to destroy any data beyond rescue, because all data are flushed to disk between each query. There hasn't been a single bug report about lost data because of bugs in **MySQL**, either.

The MyISAM table handler — Beta
: This is new in **MySQL** 3.23. It's largely based on the ISAM table code but has a lot of new very useful features.

The parser and lexical analyser — Stable
: There hasn't been a single reported bug in this system for a long time.

The C client code — Stable
: No known problems. In early 3.20 releases, there were some limitations in the send/receive buffer size. As of 3.21.x, the buffer size is now dynamic up to a default of 24M.

Standard client programs — Stable
: These include `mysql`, `mysqladmin` and `mysqlshow`, `mysqldump`, and `mysqlimport`.

Basic SQL — Stable
: The basic SQL function system and string classes and dynamic memory handling. Not a single reported bug in this system.

Query optimizer — Stable

Range optimizer — Gamma

Join optimizer — Stable

Locking — Gamma
: This is very system-dependent. On some systems there are big problems using standard OS locking (`fcntl()`). In these cases, you should run the **MySQL** daemon with the `--skip-locking` flag. Problems are known to occur on some Linux systems and on SunOS when using NFS-mounted file systems.

Linux threads — Gamma
The only problem found has been with the `fcntl()` call, which is fixed by using the `--skip-locking` option to `mysqld`. Some people have reported lockup problems with the 0.5 release.

Solaris 2.5+ pthreads — Stable
We use this for all our production work.

MIT-pthreads (Other systems) — Gamma
There have been no reported bugs since 3.20.15 and no known bugs since 3.20.16. On some systems, there is a "misfeature" where some operations are quite slow (a 1/20 second sleep is done between each query). Of course, MIT-pthreads may slow down everything a bit, but index-based `SELECT` statements are usually done in one time frame so there shouldn't be a mutex locking/thread juggling.

Other thread implementions — Alpha - Beta
The ports to other systems are still very new and may have bugs, possibly in **MySQL**, but most often in the thread implementation itself.

`LOAD DATA ...`, `INSERT ... SELECT` — **Stable**
Some people have thought they have found bugs here, but these usually have turned out to be misunderstandings. Please check the manual before reporting problems!

`ALTER TABLE` — **Stable**
Small changes in 3.22.12.

DBD — Stable
Now maintained by Jochen Wiedmann `wiedmann@neckar-alb.de`. Thanks!

`mysqlaccess` — **Stable**
Written and maintained by Yves Carlier `Yves.Carlier@rug.ac.be`. Thanks!

`GRANT` — **Gamma**
Big changes made in **MySQL** 3.22.12.

MyODBC (uses ODBC SDK 2.5) — Beta
It seems to work well with some programs.

TcX provides email support for paying customers, but the **MySQL** mailing list usually provides answers to common questions. Bugs are usually fixed right away with a patch; for serious bugs, there is almost always a new release.

1.6 Year 2000 compliance

MySQL itself has no problems with Year 2000 (Y2K) compliance:

- **MySQL** uses Unix time functions and has no problems with dates until `2069`; all 2-digit years are regarded to be in the range `1970` to `2069`, which means that if you store `01` in a `year` column, **MySQL** treats it as `2001`.

- All **MySQL** date functions are stored in one file '`sql/time.cc`' and coded very carefully to be year 2000-safe.
- In **MySQL** 3.22 and later versions, the new `YEAR` column type can store years `0` and `1901` to `2155` in 1 byte and display them using 2 or 4 digits.

You may run into problems with applications that use **MySQL** in a way that is not Y2K-safe. For example, many old applications store or manipulate years using 2-digit values (which are ambiguous) rather than 4-digit values. This problem may be compounded by applications that use values such as `00` or `99` as "missing" value indicators.

Unfortunately, these problems may be difficult to fix, since different applications may be written by different programmers, each of whom may use a different set of conventions and date-handling functions.

Here is a simple demonstration illustrating that **MySQL** doesn't have any problems with dates until the year 2030!

```
mysql> DROP TABLE IF EXISTS y2k;
mysql> CREATE TABLE y2k (date date, date_time datetime, time_stamp timestamp);
mysql> INSERT INTO y2k VALUES ("1998-12-31","1998-12-31 23:59:59",19981231235959);
mysql> INSERT INTO y2k VALUES ("1999-01-01","1999-01-01 00:00:00",19990101000000);
mysql> INSERT INTO y2k VALUES ("1999-09-09","1999-09-09 23:59:59",19990909235959);
mysql> INSERT INTO y2k VALUES ("2000-01-01","2000-01-01 00:00:00",20000101000000);
mysql> INSERT INTO y2k VALUES ("2000-02-28","2000-02-28 00:00:00",20000228000000);
mysql> INSERT INTO y2k VALUES ("2000-02-29","2000-02-29 00:00:00",20000229000000);
mysql> INSERT INTO y2k VALUES ("2000-03-01","2000-03-01 00:00:00",20000301000000);
mysql> INSERT INTO y2k VALUES ("2000-12-31","2000-12-31 23:59:59",20001231235959);
mysql> INSERT INTO y2k VALUES ("2001-01-01","2001-01-01 00:00:00",20010101000000);
mysql> INSERT INTO y2k VALUES ("2004-12-31","2004-12-31 23:59:59",20041231235959);
mysql> INSERT INTO y2k VALUES ("2005-01-01","2005-01-01 00:00:00",20050101000000);
mysql> INSERT INTO y2k VALUES ("2030-01-01","2030-01-01 00:00:00",20300101000000);
mysql> INSERT INTO y2k VALUES ("2050-01-01","2050-01-01 00:00:00",20500101000000);
mysql> SELECT * FROM y2k;
+------------+---------------------+----------------+
| date       | date_time           | time_stamp     |
+------------+---------------------+----------------+
| 1998-12-31 | 1998-12-31 23:59:59 | 19981231235959 |
| 1999-01-01 | 1999-01-01 00:00:00 | 19990101000000 |
| 1999-09-09 | 1999-09-09 23:59:59 | 19990909235959 |
| 2000-01-01 | 2000-01-01 00:00:00 | 20000101000000 |
| 2000-02-28 | 2000-02-28 00:00:00 | 20000228000000 |
| 2000-02-29 | 2000-02-29 00:00:00 | 20000229000000 |
| 2000-03-01 | 2000-03-01 00:00:00 | 20000301000000 |
| 2000-12-31 | 2000-12-31 23:59:59 | 20001231235959 |
| 2001-01-01 | 2001-01-01 00:00:00 | 20010101000000 |
| 2004-12-31 | 2004-12-31 23:59:59 | 20041231235959 |
| 2005-01-01 | 2005-01-01 00:00:00 | 20050101000000 |
| 2030-01-01 | 2030-01-01 00:00:00 | 20300101000000 |
| 2050-01-01 | 2050-01-01 00:00:00 | 00000000000000 |
+------------+---------------------+----------------+
```

```
13 rows in set (0.00 sec)
```

This shows that the `DATE` and `DATETIME` types are will not give any problems with future dates (they handle dates until the year 9999).

The `TIMESTAMP` type, that is used to store the current time, has a range up to only `2030-01-01`. `TIMESTAMP` has a range of `1970` to `2030` on 32-bit machines (signed value). On 64-bit machines it handles times up to `2106` (unsigned value).

Even though **MySQL** is Y2K-compliant, it is your responsibility to provide unambiguous input. See Section 7.2.3.1 [Y2K issues], page 126 for **MySQL**'s rules for dealing with ambiguous date input data (data containing 2-digit year values).

1.7 General SQL information and tutorials

This book has been recommended by a several people on the **MySQL** mailing list:

```
Judith S. Bowman, Sandra L. Emerson and Marcy Darnovsky
The Practical SQL Handbook: Using Structured Query Language
Second Edition
Addison-Wesley
ISBN 0-201-62623-3
http://www.awl.com
```

This book has also received some recommendations on the mailing list:

```
Martin Gruber
Understanding SQL
ISBN 0-89588-644-8
Publisher Sybex 510 523 8233
Alameda, CA USA
```

A SQL tutorial is available on the net at `http://www.geocities.com/SiliconValley/Vista/2207/sql1.h`

SQL in 21 Tagen (online book in German language): `http://www.mut.de/leseecke/buecher/sql/inhalt`

1.8 Useful MySQL-related links

Applications that support MySQL

- SupportWizard; Interactive helpdesk on the web (This product includes a licensed copy of MySQL)
- Right Now Web; Web automation for customer service
- Bazaar; Interactive Discussion Forums with web interface

SQL Clients

- **MySQL** Editor/Utility for MS Windows Platforms.
- KDE MySQL client
- Kiosk; a MySQL client for database management. Written in Perl. Will be a part of Bazaar.

Web development tools that support MySQL

- PHP: A server-side HTML-embedded scripting language
- The Midgard Application Server; a powerful Web development environment based on MySQL and PHP
- SmartWorker is a platform for web application development
- XSP: e(X)tendible (s)erver (p)ages and is a HTML embedded tag language written in Java (previously known as XTAGS)
- Platform independent ASP from Chili!Soft
- **MySQL** + PHP demos
- ForwardSQL: HTML interface to manipulate **MySQL** databases
- WWW-SQL: Display database information
- Minivend: A Web shopping cart
- HeiTML: A server-side extension of HTML and a 4GL language at the same time
- Metahtml: A Dynamic Programming Language for WWW Applications
- VelocityGen for Perl and TCL
- Hawkeye Internet Server Suite
- Network Database Connection For Linux
- WDBI: Web browser as a universal front end to databases which supports **MySQL** well.
- WebGroove Script: HTML compiler and server-side scripting language
- A server-side web site scripting language
- How to use **MySQL** with Coldfusion on Solaris
- Calistra's ODBC **MySQL** Administrator
- Webmerger This CGI tool interprets files and generates dynamic output based on a set of simple tags. Ready-to-run drivers for **MySQL** and PostgreSQL through ODBC.
- PHPclub. Tips and tricks for PHP
- MySQL and Perl Scripts
- The Widgetchuck; Web Site Tools and Gadgets

Databse design tools with MySQL support

- "Dezign for databases" is a database development tool using an rick> entity relationship diagram (ERD).

Web servers with MySQL tools

- An Apache authentication module
- The Roxen Challenger Web server

Extension for other programs

A Delphi interface to **MySQL**. With source code. By Matthias Fichtner.

- **TmySQL**; A library to use **MySQL** with Delphi
- Delphi TDataset-component
- Support for BIND (The Internet Domain Name Server)
- Sendmail extensions using MySQL

ODBC related links

- Popular iODBC Driver Manager now available in Open Source format
- The FreeODBC Pages

API related links

- www.jppp.com Partially implemented TDataset-compatible components for **MySQL**.
- qpopmysql A patch to allow POP3 authentication from a **MySQL** database. There's also a link to someone who wrote a patch for Procmail to allow any MTA to deliver to users in a **MySQL** database.
- Visual Basic class generator for Active X
- Client libraries for the Macintosh
- **MySQL** binding to Free Pascal

Other MySQL-related links

- Registry of Web providers who support **MySQL** Links about using **MySQL** in Japan/Asia
- Commercial Web defect tracking system
- PTS: Project Tracking System
- Job and software tracking system
- ExportSQL: A script to export data from Access95+
- SAL (Scientific Applications on Linux) **MySQL** entry
- A consulting company which mentions **MySQL** in the right company
- PMP Computer Solutions. Database developers using **MySQL** and `mSQL`
- Airborne Early Warning Association
- **MySQL** UDF Registry
- Y2K tester

SQL and database interfaces

- KMySQL KMySQL is a database client for KDE that primarily supports **MySQL**.
- The JDBC database access API

- Patch for `mSQL` TCL
- EasySQL: An ODBC-like driver manager
- A REXX interface to SQL databases
- TCL interface

Examples of MySQL use

- Little6 Inc An online contract and job finding site that is powered by **MySQL**, PHP3 and Linux.
- DELECis A tool which makes it very easy to create an automatically generated table documentation. They have used **MySQL** as an example.
- Steve Fambro Uses **MySQL** and webmerger. There is an employee database, and a license plate database with all of the registered Utah vehicles (over 1.2 million). The License plate field is indexed.....so the *searches* are instantaneous.
- World Records A search engine for information about music that uses **MySQL** and PHP.
- Examples using **MySQL**; (check Top 10)
- A Contact Database using **MySQL** and PHP
- Web based interface and Community Calender with PHP
- Perl package to generate html from a SQL table structure and for generating SQL statements from an html form.
- Basic telephone database using `DBI/DBD`.
- Installing new Perl modules that require locally installed modules
- JDBC examples by Daniel K. Schneider
- SQL BNF
- Object Oriented Concepts Inc; CORBA applications with examples in source
- DBWiz; Includes an example of how to manage own cursors in VB
- Pluribus Pluribus, is a free search engine that learns to improve the quality of its results over time. Pluribus works by recording which pages a user prefers among those returned for a query. A user votes for a page by selecting it; Pluribus then uses that knowledge to improve the quality of the results when someone else submits the same (or similar) query. Uses PHP and **MySQL**.
- Stopbit A technology news site using **MySQL** and PHP
- Example scripts at Jokes2000
- **MySQL**-perl tutorial
- PHP/MySQL Tutorial
- FutureForum Web Discussion Software
- `http://www.linuxsupportline.com/~kalendar/ KDE based calendar manager` The calendar manager has both single user (file based) and multi user (**MySQL** database) support.
- Example of storing/retrieving images with **MySQL** and CGI
- Online shopping cart system

General database links

- Database Jump Site
- Homepage of the webdb-l (Web Databases) mailing list.
- Perl `DBI/DBD` modules homepage
- Cygwin tools (MySQL +Apache + PHP under Win32
- dbasecentral.com; Development and distribution of powerful and easy-to-use database applications and systems.
- Tek-Tips Forums Tek-Tips Forums are 800+ independent peer-to-peer non-commercial support forums for Computer Professionals. Features include automatic e-mail notification of responses, a links library, and member confidentiality guaranteed.

There are also many web pages that use **MySQL**. See Appendix A [Users], page 382. Send any additions to this list to `webmaster@mysql.com`. We now require that you show a **MySQL** logo somewhere (It is okay to have it on a "used tools" page or something similar) to be added.

2 MySQL mailing lists and how to ask questions or report errors (bugs)

2.1 The MySQL mailing lists

To subscribe to the main **MySQL** mailing list, send a message to the electronic mail address mysql-subscribe@lists.mysql.com.

To unsubscribe from the main **MySQL** mailing list, send a message to the electronic mail address mysql-unsubscribe@lists.mysql.com.

Only the address to which you send your messages is significant. The subject line and the body of the message are ignored.

If your reply address is not valid, you can specify your address explicitly. Adding a hyphen to the subscribe or unsubscribe command word, followed by your address with the '@' character in your address replaced by a '='. For example, to subscribe john@host.domain, send a message to mysql-subscribe-john=host.domain@lists.mysql.com.

Mail to mysql-subscribe@lists.mysql.com or mysql-unsubscribe@lists.mysql.com is handled automatically by the ezmlm mailing list processor. Information about ezmlm is available at the ezmlm Website.

To post a message to the list itself, send your message to mysql@lists.mysql.com. However, please *do not* send mail about subscribing or unsubscribing to mysql@lists.mysql.com, since any mail sent to that address is distributed automatically to thousands of other users.

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

The following **MySQL** mailing lists exist:

announce
: This is for announcement of new versions of **MySQL** and related programs. This is a low volume list that we think all **MySQL** users should be on.

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

mysql-digest
: The mysql list in digest form. That means you get all individual messages, sent as one large mail message once a day.

java
: Discussion about **MySQL** and Java. Mostly about the JDBC drivers.

java-digest
: A digest version of the java list.

win32
: All things concerning **MySQL** on Microsoft operating systems such as Windows NT.

win32-digest
: A digest version of the win32 list.

`myodbc` All things concerning connecting to **MySQL** with ODBC.

`myodbc-digest`
A digest version of the `myodbc` list.

`msql-mysql-modules`
A list about the Perl support in **MySQL**.

`msql-mysql-modules-digest`
A digest version of the `msql-mysql-modules` list.

`developer`
A list for people who work on the **MySQL** code.

`developer-digest`
A digest version of the `developer` list.

You subscribe or unsubscribe to all lists in the same way as described above. In your subscribe or unsubscribe message, just put the appropriate mailing list name rather than `mysql`. For example, to subscribe to or unsubscribe from the `myodbc` list, send a message to `myodbc-subscribe@lists.mysql.com` or `myodbc-unsubscribe@lists.mysql.com`.

2.2 Asking questions or reporting bugs

Before posting a bug report or question, please do the following:

- Start by searching the **MySQL** online manual at:

  `http://www.mysql.com/Manual_chapter/manual_toc.html`

  We try to keep the manual up to date by updating it frequently with solutions to newly found problems!
- Search the **MySQL** mailing list archives:

  `http://www.mysql.com/doc.html`
- You can also use `http://www.mysql.com/search.html` to search all the web pages (including the manual) that are located at `http://www.mysql.com/`.

If you can't find an answer in the manual or the archives, check with your local **MySQL** expert. If you still can't find an answer to your question, go ahead and read the next section about how to send mail to `mysql@lists.mysql.com`.

2.3 How to report bugs or problems

Writing a good bug report takes patience, but doing it right the first time saves time for us and for you. This section will help you write your report correctly so that you don't waste your time doing things that may not help us much or at all.

We encourage everyone to use the `mysqlbug` script to generate a bug report (or a report about any problem), if possible. `mysqlbug` can be found in the '`scripts`' directory in the source distribution, or, for a binary distribution, in the '`bin`' directory under your **MySQL** installation directory. If you are unable to use `mysqlbug`, you should still include all the necessary information listed in this section.

The `mysqlbug` script helps you generate a report by determining much of the following information automatically, but if something important is missing, please include it with your message! Please read this section carefully and make sure that all the information described here is included in your report.

Remember that it is possible to respond to a message containing too much information, but not to one containing too little. Often people omit facts because they think they know the cause of a problem and assume that some details don't matter. A good principle is: if you are in doubt about stating something, state it! It is a thousand times faster and less troublesome to write a couple of lines more in your report than to be forced to ask again and wait for the answer because you didn't include enough information the first time.

The most common errors are that people don't indicate the version number of the **MySQL** distribution they are using, or don't indicate what platform they have **MySQL** installed on (including the platform version number). This is highly relevant information and in 99 cases out of 100 the bug report is useless without it! Very often we get questions like "Why doesn't this work for me?" and then we find that the feature requested wasn't implemented in that **MySQL** version, or that a bug described in a report has been fixed already in newer **MySQL** versions. Sometimes the error is platform dependent; in such cases, it is next to impossible to fix anything without knowing the operating system and the version number of the platform.

Remember also to provide information about your compiler, if it is related to the problem. Often people find bugs in compilers and think the problem is **MySQL** related. Most compilers are under development all the time and become better version by version, too. To determine whether or not your problem depends on your compiler, we need to know what compiler is used. Note that every compiling problem should be regarded as a bug report and reported accordingly.

It is most helpful when a good description of the problem is included in the bug report. That is, a good example of all the things you did that led to the problem and the problem itself exactly described. The best reports are those that include a full example showing how to reproduce the bug or problem.

If a program produces an error message, it is very important to include the message in your report! If we try to search for something from the archives using programs, it is better that the error message reported exactly matches the one that the program produces. (Even the case sensitivity should be observed!) You should never try to remember what the error message was; instead, copy and paste the entire message into your report!

Please include the following information in your report:

- The version number of the **MySQL** distribution you are using (for example, **MySQL** 3.22.22). You can find out which version you are running by executing `mysqladmin version`. `mysqladmin` can be found in the '`bin`' directory under your **MySQL** installation directory.
- The manufacturer and model of the machine you are working on.
- The operating system name and version. For most operating systems, you can get this information by executing the Unix command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.

- If you are using a source distribution of **MySQL**, the name and version number of the compiler used is needed. If you have a binary distribution, the distribution name is needed.
- If the problem occurs during compilation, include the exact error message(s) and also a few lines of context around the offending code in the file where the error occurred.
- If any database table is related to the problem, include the output from `mysqldump --no-data db_name tbl_name1 tbl_name2 ...` This is very easy to do and is a powerful way to get information about any table in a database that will help us create a situation matching the one you have.
- For speed-related bugs or problems with `SELECT` statements, you should always include the output of `EXPLAIN SELECT ...`, and at least the number of rows that the `SELECT` statement produces. The more information you give about your situation, the more likely it is that someone can help you! For example, the following is an example of a very good bug report (it should of course be posted with the `mysqlbug` script):

  Example run under the `mysql` command line tool:

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

- If a bug or problem occurs while running **MySQL**, try to provide an input script that will reproduce the anomaly. This script should include any necessary source files. The more closely the script can reproduce your situation, the better.

  If you can't provide a script, you should at least include the output from `mysqladmin variables extended-status processlist` in your mail to provide some information of how your system is performing!
- If you think that **MySQL** produces a strange result from a query, include not only the result, but also your opinion of what the result should be and an account describing the basis for your opinion.
- When giving an example of the problem, it's better to use the variable names, table names, etc., that exist in your actual situation than to come up with new names. The problem could be related to the name of a variable, table, etc.! These cases are rare, perhaps, but it is better to be safe than sorry. After all, it should be easier for you to provide an example that uses your actual situation and it is by all means better for us. In case you have data you don't want to show to others, you can use `ftp` to transfer it to `ftp://www.mysql.com/pub/mysql/secret/`. If the data are really top secret and you don't want to show them even to us, then go ahead and provide an example using other names, but please regard this as the last choice.
- Include all the options given to the relevant programs, if possible. For example, indicate the options that you use when you start the `mysqld` daemon and that you use to run

any **MySQL** client programs. The options to programs like `mysqld` and `mysql`, and to the `configure` script are often keys to answers and very relevant! It is never a bad idea to include them anyway! If you use any modules, such as Perl or PHP, please include the version number(s) of those as well.

- If you can't produce a test case in a few rows, or if the test table is too big to be mailed to the mailing list (more than 10 rows), you should dump your tables using `mysqldump` and create a '`README`' file that describes your problem.

  Create a compressed archive of your files using `tar` and `gzip` or `zip`, and use `ftp` to transfer the archive to `ftp://www.mysql.com/pub/mysql/secret/`. Then send a short description of the problem to `mysql@lists.mysql.com`.
- If your question is related to the privilege system, please include the output of `mysqlaccess`, the output of `mysqladmin reload` and all the error messages you get when trying to connect! When you test your privileges, you should first run `mysqlaccess`. After this, execute `mysqladmin reload version`, and last you should try to connect with the program that gives you trouble. `mysqlaccess` can be found in the '`bin`' directory under your **MySQL** installation directory.
- If you have a patch for a bug, that is good. But don't assume the patch is all we need or that we will use it even if you don't provide some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all; if so, we can't use it.

  If we can't verify exactly what the patch is meant for, we won't use it. Test cases will help us here. Show that the patch will handle all the situations that may occur. If we find a borderline case (even a rare one) where the patch won't work, the patch may be useless.
- Guesses about what the bug is, why it occurs, or what it depends on, are usually wrong. Even we can't guess such things without first using a debugger to determine the real cause of a bug.
- Indicate in your mail message that you have checked the reference manual and mail archive so others know that you have tried to solve your problem yourself.
- If you get a `parse error`, please check your syntax closely! If you can't find something wrong with it, it's extremely likely that your current version of **MySQL** doesn't support the query you are using. If you are using the current version and the manual at `http://www.mysql.com/doc.html` doesn't cover the syntax you are using, **MySQL** doesn't support your query. In this case, your only options are to implement the syntax yourself or email `mysql-licensing@mysql.com` and ask for an offer to implement it!

  If the manual covers the syntax you are using, but you have an older version of **MySQL**, you should check the **MySQL** change history to see when the syntax was implemented. See Appendix D [News], page 400. In this case, you have the option of upgrading to a newer version of **MySQL**.
- If you have a problem such that your data appears corrupt or you get errors when you access some particular table, you should first check and then try repairing your tables with `isamchk`. See Chapter 13 [Maintenance], page 280.
- If you often get corrupted tables you should try to find out when and why this happens! In this case, the '`mysql-data-directory/'hostname'.err`' file may contain some in-

formation about what happened. Please include any relevant information from this file in your bug report! Normally `mysqld` should **NEVER** crash a table if nothing killed it in the middle of an update! If you can find the source of why `mysqld` dies, it's much easier for us to provide you with a fix for the problem!

- If possible, download the most recent version of **MySQL** and check whether or not it solves your problem. All versions of **MySQL** are thoroughly tested and should work without problems! We believe in making everything as backward compatible as possible and you should be able to switch **MySQL** versions in minutes! See Section 4.3 [Which version], page 33.

If you are a support customer, please cross-post the bug report to `mysql-support@mysql.com` for higher priority treatment, as well as to the appropriate mailing list to see if someone else has experienced (and perhaps solved) the problem.

For information on reporting bugs in **MyODBC**, see Section 16.2 [ODBC Problems], page 302.

For solutions to some common problems, see See Chapter 18 [Problems], page 307.

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

2.4 Guidelines for answering questions on the mailing list

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

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

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

3 MySQL licensing and support

This chapter describes **MySQL** licensing and support arrangements, including:

- Our licensing policies for non-Microsoft and Microsoft operating systems
- The copyrights under which **MySQL** is distributed (see Section 3.2 [Copyright], page 21)
- Sample situations illustrating when a license is required (see Section 3.4 [Licensing examples], page 22)
- Licensing and support costs (see Section 3.5 [Cost], page 24), and support benefits (see Section 3.6 [Support], page 26)

3.1 MySQL licensing policy

The formal terms of the license for non-Microsoft operating systems such as Unix or OS/2 are specified in Appendix J [Public license], page 454. Basically, our licensing policy is as follows:

- For normal internal use, **MySQL** generally costs nothing. You do not have to pay us if you do not want to.
- A license is required if:
  - You sell the **MySQL** server directly or as a part of another product or service
  - You charge for installing and maintaining a **MySQL** server at some client site
  - You include **MySQL** in a distribution that is non redistributable and you charge for some part of that distribution
- For circumstances under which a **MySQL** license is required, you need a license per machine that runs the `mysqld` server. However, a multiple-CPU machine counts as a single machine, and there is no restriction on the number of **MySQL** servers that run on one machine, or on the number of clients concurrently connected to a server running on that machine!
- You do not need a license to include client code in commercial programs. The client access part of **MySQL** is in the public domain. The `mysql` command line client includes code from the `readline` library that is under the GNU Public License.
- For customers who have purchased 10 licenses or a high enough level of support, we provide additional functionality. Currently, this means we provide the `pack_isam` utility for creating fast compressed read-only databases. (The server includes support for reading such databases but not the packing tool used to create them.) If support agreements generate sufficient revenue, we will probably release this tool under the same license as the **MySQL** server.
- If your use of **MySQL** does not require a license, but you like **MySQL** and want to encourage further development, you are certainly welcome to purchase a license anyway.
- If you use **MySQL** in a commercial context such that you profit by its use, we ask that you further the development of **MySQL** by purchasing some level of support. We feel that if **MySQL** helps your business, it is reasonable to ask that you help **MySQL**. (Otherwise, if you ask us support questions, you are not only using for free something into which we've put a lot a work, you're asking us to provide free support, too.)

For use under Microsoft operating systems (Win95/Win98/WinNT), you need a **MySQL** license after a trial period of 30 days, with the exception that licenses may be obtained upon request at no cost for educational use or for university- or government-sponsored research settings. See Appendix K [Win license], page 457. A shareware version of **MySQL**-Win32 that you can try before buying is available at `http://www.mysql.com/mysql_w32.htmy`. After you have paid, you will get a password that will enable you to access the newest **MySQL**-Win32 version.

If you have any questions as to whether or not a license is required for your particular use of **MySQL**, please contact us. See Section 3.5.2 [Contact information], page 26.

If you require a **MySQL** license, the easiest way to pay for it is to use the license form at TcX's secure server at `https://www.mysql.com/license.htmy`. Other forms of payment are discussed in Section 3.5.1 [Payment information], page 25.

3.2 Copyrights used by MySQL

There are several different copyrights on the **MySQL** distribution:

1. The **MySQL**-specific source needed to build the `mysqlclient` library and programs in the '`client`' directory is in the public domain. Each file that is in the public domain has a header which clearly states so. This includes everything in the '`client`' directory and some parts of the `mysys`, `mystring` and `dbug` libraries.
2. Some small parts of the source (GNU `getopt`) are covered by the "GNU LIBRARY LIBRARY GENERAL PUBLIC LICENSE". See the '`mysys/COPYING.LIB`' file.
3. Some small parts of the source (GNU `readline`) are covered by the "GNU GENERAL PUBLIC LICENSE". See the '`readline/COPYING`' file.
4. Some parts of the source (the `regexp` library) are covered by a Berkeley style copyright.
5. The other source needed for the **MySQL** server on non-Microsoft platforms is covered by the "MySQL FREE PUBLIC LICENSE", which is based on the "Aladdin FREE PUBLIC LICENSE." See Appendix J [Public license], page 454. When running **MySQL** on any Microsoft operating system, other licensing applies.

The following points set forth the philosophy behind our copyright policy:

- The SQL client library should be totally free so that it can be included in commercial products without limitations.
- People who want free access to the software into which we have put a lot of work can have it, so long as they do not try to make money directly by distributing it for profit.
- People who want the right to keep their own software proprietary, but also want the value from our work, can pay for the privilege.
- That means normal in-house use is FREE. But if you use **MySQL** for something important to you, you may want to help further its development by purchasing a license or a support contract. See Section 3.6 [Support], page 26.

3.2.1 Possible future copyright changes

We may choose to distribute older versions of **MySQL** with the GPL in the future. However, these versions will be identified as **GNU MySQL**. Also, all copyright notices in the relevant files will be changed to the GPL.

3.3 Distributing MySQL commercially

This section is a clarification of the license terms that are set forth in the "MySQL FREE PUBLIC LICENSE" (FPL). See Appendix J [Public license], page 454.

MySQL may be **used** freely, including by commercial entities for evaluation or unsupported internal use. However, **distribution** for commercial purposes of **MySQL**, or anything containing or derived from **MySQL** in whole or in part, requires a written commercial license from TcX AB, the sole entity authorized to grant such licenses.

You may not include **MySQL** "free" in a package containing anything for which a charge is being made, except as noted below.

The intent of the exception provided in the second clause of the license is to allow commercial organizations operating an FTP server or a bulletin board to distribute **MySQL** freely from it, provided that:

1. The organization complies with the other provisions of the FPL, which include among other things a requirement to distribute the full source code of **MySQL** and of any derived work, and to distribute the FPL itself along with **MySQL**;
2. The only charge for downloading **MySQL** is a charge based on the distribution service and not one based on the content of the information being retrieved (i.e., the charge would be the same for retrieving a random collection of bits of the same size);
3. The server or BBS is accessible to the general public, i.e., the phone number or IP address is not kept secret, and anyone may obtain access to the information (possibly by paying a subscription or access fee that is not dependent on or related to purchasing anything else).

If you want to distribute software in a commercial context that incorporates **MySQL** and you do **not** want to meet these conditions, you should contact TcX AB to find out about commercial licensing, which involves a payment. The only ways you legally can distribute **MySQL** or anything containing **MySQL** are by distributing **MySQL** under the requirements of the FPL, or by getting a commercial license from TcX AB.

3.4 Example licensing situations

This section describes some situations illustrating whether or not you must license the **MySQL** server. Generally these examples involve providing **MySQL** as part of a product or service that you are selling to a customer, or requiring that **MySQL** be used in conjunction with your product. In such cases, it is your responsibility to obtain a license for the customer if one is necessary. (This requirement is waived if your customer already has a **MySQL** license. But the seller must send customer information and the license number to TcX, and the license must be a full license, not an OEM license.)

Note that a single **MySQL** license covers any number of CPUs/users/customers/`mysqld` servers on a machine!

3.4.1 Selling products that use MySQL

To determine whether or not you need a **MySQL** license when selling your application, you should ask whether the proper functioning of your application is contingent on the use of **MySQL** and whether you include **MySQL** with your product. There are several cases to consider:

- Does your application require **MySQL** to function properly?

  If your product requires **MySQL**, you need a license for any machine that runs the `mysqld` server. For example, if you've designed your application around **MySQL**, then you've really made a commercial product that requires the engine, so you need a license.

  If your application does not require **MySQL**, you need not obtain a license. For example, if **MySQL** just added some new optional features to your product (such as adding logging to a database if **MySQL** is used rather than logging to a text file), it should fall within normal use, and a license would not be required.

  In other words, you need a license if you sell a product designed specifically for use with **MySQL** or that requires the **MySQL** server to function at all. This is true whether or not you provide **MySQL** for your client as part of your product distribution.

  It also depends on what you're doing for the client. Do you plan to provide your client with detailed instructions on installing **MySQL** with your software? Then your product may be contingent on the use of **MySQL**; if so, you need to buy a license. If you are simply tying into a database that you expect already to have been installed by the time your software is purchased, then you probably don't need a license.
- Do you include **MySQL** in a distribution and charge for that distribution?

  If you include **MySQL** with a distribution that you sell to customers, you will need a license for any machine that runs the `mysqld` server, because in this case you are selling a system that includes **MySQL**.

  This is true whether the use of **MySQL** with your product is required or optional.
- Do you neither require for your product nor include **MySQL** with it?

  Suppose you want to sell a product that is designed generally to use "some database" and that can be configured to use any of several supported alternative database systems (**MySQL**, PostgreSQL, or something else). That is, your product does not not require **MySQL**, but can support any database with a base level of functionality, and you don't rely on anything that only **MySQL** supports. Does one of you owe us money if your customer actually does choose to use **MySQL**?

  In this case, if you don't provide, obtain or set up **MySQL** for the customer should the customer decide to use it, neither of you need a license. If you do perform that service, see Section 3.4.2 [**MySQL** services], page 23.

3.4.2 Selling MySQL-related services

If you perform **MySQL** installation on a client's machine and any money changes hands for the service (directly or indirectly), then you must buy a **MySQL** license.

If you sell an application for which **MySQL** is not strictly required but can be used, a license may be indicated, depending on how **MySQL** is set up. Suppose your product neither requires **MySQL** nor includes it in your product distribution, but can be configured to use **MySQL** for those customers who so desire. (This would be the case, for example, if your product can use any of a number of database engines.)

If the customer obtains and installs **MySQL**, no license is needed. If you perform that service for your customer, then a license is needed because then you are selling a service that includes **MySQL**.

3.4.3 ISP MySQL services

Internet Service Providers (ISPs) often host **MySQL** servers for their customers.

If you are an ISP that allows customers to install and administer **MySQL** for themselves on your machine with no assistance from you, neither you nor your customer need a **MySQL** license.

If you charge for **MySQL** installation and administrative support as part of your customer service, then you need a license because you are selling a service that includes **MySQL**.

3.4.4 Running a web server using MySQL

If you use **MySQL** in conjunction with a web server, you don't have to pay for a license.

This is true even if you run a commercial web server that uses **MySQL**, since you are not selling **MySQL** itself. However, in this case we would like you to purchase **MySQL** support, because **MySQL** is helping your enterprise.

3.5 MySQL licensing and support costs

Our current license prices are shown below. All prices are in US Dollars. If you pay by credit card, the currency is EURO (European Union Euro) so the prices will differ slightly.

Number of licenses	Price per copy	Total
1	US $200	US $200
10 pack	US $150	US $1500
50 pack	US $120	US $6000

For high volume (OEM) purchases, the following prices apply:

Number of licenses	Price per copy	Minimum at one time	Minimum payment
100-999	US $40	100	US $4000
1000-2499	US $25	200	US $5000
2500-4999	US $20	400	US $8000

For OEM purchases, you must act as the middle-man for eventual problems or extension requests from your users. We also require that OEM customers have at least an extended email support contract.

If you have a low-margin high-volume product, you can always talk to us about other terms (for example, a percent of the sale price). If you do, please be informative about your product, pricing, market and any other information that may be relevant.

After buying 10 **MySQL** licenses, you will get a personal copy of the `pack_isam` utility. You are not allowed to redistribute this utility but you can distribute tables packed with it.

A full-price license is not a support agreement and includes very minimal support. This means that we try to answer any relevant question. If the answer is in the documentation, we will direct you to the appropriate section. If you have not purchased a license or support, we probably will not answer at all.

If you discover what we consider a real bug, we are likely to fix it in any case. But if you pay for support we will notify you about the fix status instead of just fixing it in a later release.

More comprehensive support is sold separately. Descriptions of what each level of support includes are given in Section 3.6 [Support], page 26. Costs for the various types of commercial support are shown below. Support level prices are in EURO (European Union Euro). One EURO is about 1.17 USD.

Type of support	**Cost per year**
Basic email support	EURO 170
Extended email support	EURO 1000
Login support	EURO 2000
Extended login support	EURO 5000

You may upgrade from any lower level of support to a higher level of support for the difference between the prices of the two support levels.

3.5.1 Payment information

Currently we can take SWIFT payments, cheques or credit cards.

Payment should be made to:

```
Postgirot Bank AB
105 06 STOCKHOLM, SWEDEN

TCX DataKonsult AB
BOX 6434
11382 STOCKHOLM, SWEDEN

SWIFT address: PGSI SESS
Account number: 96 77 06 - 3
```

Specify: license and/or support and your name and email address.

In Europe and Japan you can use EuroGiro (that should be less expensive) to the same account.

If you want to pay by cheque, make it payable to "Monty Program KB" and mail it to the address below:

```
TCX DataKonsult AB
BOX 6434
11382 STOCKHOLM, SWEDEN
```

If you want to pay by credit card over the Internet, you can use TcX's secure license form.

You can also print a copy of the license form, fill it in and send it by fax to:

+46-8-729 69 05

If you want us to bill you, you can use the license form and write "bill us" in the comment field. You can also mail a message to sales@mysql.com (**not** mysql@lists.mysql.com!) with your company information and ask us to bill you.

3.5.2 Contact information

For commercial licensing, or if you have any questions about any of the information in this section, please contact the **MySQL** licensing team. The much preferred method is by E-Mail to mysql-licensing@mysql.com. Fax is also possible but handling of these may take much longer (Fax +46-8-729 69 05).

```
David Axmark
Detron HB
Kungsgatan 65 B
753 21 UPPSALA
SWEDEN
Voice Phone +46-18-10 22 80     (Timezone GMT+1. Swedish and English spoken)
```

3.6 Types of commercial support

3.6.1 Basic email support

Basic email support is a very inexpensive support option and should be thought of more as a way to support our development of **MySQL** than as a real support option.

At this support level, the **MySQL** mailing lists are the preferred means of communication. Questions normally should be mailed to the primary mailing list (mysql@lists.mysql.com) or one of the other regular lists (for example, mysql-win32@lists.mysql.com for Windows-related **MySQL** questions), as someone else already may have experienced and solved the problem you have. See Section 2.2 [Asking questions], page 15.

However, by purchasing basic email support, you also have access to the support address mysql-support@mysql.com, which is not available as part of the minimal support that you get by purchasing a **MySQL** license. This means that for especially critical questions, you can cross-post your message to mysql-support@mysql.com. (If the message contains sensitive data, you should post only to mysql-support@mysql.com.)

REMEMBER! to ALWAYS include your registration number and expiration date when you send a message to mysql-support@mysql.com.

Basic email support includes the following types of service:

- If your question is already answered in the manual, we will inform you of the correct section in which you can find the answer. If the answer is not in the manual, we will point you in the right direction to solve your problem.
- We guarantee a timely answer for your email messages. We can't guarantee that we can solve any problem, but at least you will receive an answer if we can contact you by email.

- We will help with unexpected problems when you install **MySQL** from a binary distribution on supported platforms. This level of support does not cover installing **MySQL** from a source distribution. "Supported" platforms are those for which **MySQL** is known to work. See Section 4.2 [Which OS], page 32.
- We will help you with bugs and missing features. Any bugs that are found are fixed for the next **MySQL** release. If the bug is critical for you, we will mail you a patch for it as soon the bug is fixed. Critical bugs always have the highest priority for us, to ensure that they are fixed as soon as possible.
- Your suggestions for the further development of **MySQL** will be taken into consideration. By taking email support you have already helped the further development of **MySQL**. If you want to have more input, upgrade to a higher level of support.
- If you want us to help optimize your system, you must upgrade to a higher level of support.

3.6.2 Extended email support

Extended email support includes everything in basic email support with these additions:

- Your email will be dealt with before mail from basic email support users and non-registered users.
- Your suggestions for the further development of **MySQL** will receive strong consideration. Simple extensions that suit the basic goals of **MySQL** are implemented in a matter of days. By taking extended email support you have already helped the further development of **MySQL**.
- We include a binary version of the `pack_isam` packing tool for creating fast compressed read-only databases (it does not support `BLOB` or `TEXT` types yet). The current server includes support for reading such databases but not the packing tool used to create them.
- Typical questions that are covered by extended email support are:
  - We will answer and (within reason) solve questions that relate to possible bugs in **MySQL**. As soon as the bug is found and corrected, we will mail a patch for it.
  - We will help with unexpected problems when you install **MySQL** from a source or binary distribution on supported platforms.
  - We will answer questions about missing features and offer hints how to work around them.
  - We will provide hints on optimizing `mysqld` for your situation.
- You are allowed to influence the priority of items on the **MySQL** TODO. This will ensure that the features you really need will be implemented sooner than they might be otherwise.

3.6.3 Login support

Login support includes everything in extended email support with these additions:

- Your email will be dealt with even before mail from extended email support users.

- Your suggestions for the further development of **MySQL** will be taken into very high consideration. Realistic extensions that can be implemented in a couple of hours and that suit the basic goals of **MySQL** will be implemented as soon as possible.
- If you have a very specific problem, we can try to log in on your system to solve the problem "in place."
- Like any database vendor, we can't guarantee that we can rescue any data from crashed tables, but if the worst happens we will help you rescue as much as possible. **MySQL** has proven itself very reliable, but anything is possible due to circumstances beyond our control (for example, if your system crashes or someone kills the server by executing a `kill -9` command).
- We will provide hints on optimizing your system and your queries.
- You are allowed to call a **MySQL** developer (in moderation) and discuss your **MySQL**-related problems.

3.6.4 Extended login support

Extended login support includes everything in login support with these additions:

- Your email has the highest possible priority.
- We will actively examine your system and help you optimize it and your queries. We may also optimize and/or extend **MySQL** to better suit your needs.
- You may also request special extensions just for you. For example:

```
mysql> select MY_CALCULATION(col_name1,col_name2) from tbl_name;
```

- We will provide a binary distribution of all important **MySQL** releases for your system, as long as we can get an account on a similar system. In the worst case, we may require access to your system to be able to create a binary distribution.
- If you can provide accommodations and pay for traveler fares, you can even get a **MySQL** developer to visit you and offer you help with your troubles. Extended login support entitles you to one personal encounter per year, but we are as always very flexible towards our customers!

4 Installing MySQL

This chapter describes how to obtain and install **MySQL**:

- For a list of sites from which you can obtain **MySQL**, see Section 4.1 [Getting **MySQL**], page 29.
- To see which platforms are supported, see Section 4.2 [Which OS], page 32.
- Several versions of **MySQL** are available, in both binary and source distributions. To determine which version and type of distribution you should use, see Section 4.4 [Many versions], page 34.
- Installation instructions for binary and source distributions are described in Section 4.6 [Installing binary], page 36, and Section 4.7 [Installing source], page 40. Each set of instructions includes a section on system-specific problems you may run into.
- For post-installation procedures, see Section 4.15 [Post-installation], page 72. These procedures apply whether you install **MySQL** using a binary or source distribution.

4.1 How to get MySQL

Check the **MySQL** home page for information about the current version and for downloading instructions.

However, the Internet connection at TcX is not so fast; we would *prefer* that you do the actual downloading from one of the mirror sites listed below.

Please report bad or out of date mirrors to `webmaster@mysql.com`.

Europe:

- Austria [Univ. of Technology/Vienna] WWW FTP
- Bulgaria [Naturella] FTP
- Croatia [HULK] WWW FTP
- Czech Republic [Masaryk University in Brno] WWW FTP
- Denmark [Ake] WWW
- Denmark [SunSITE] WWW FTP
- Estonia [OKinteractive] WWW

- France [minet] WWW
- Finland [EUnet] WWW
- Finland [clinet] FTP
- Germany [Bonn University, Bonn] WWW FTP
- Germany [Wolfenbuettel] WWW FTP
- Germany [Staufen] WWW
- Germany [Cable & Wireless] FTP
- Greece [NTUA, Athens] WWW FTP
- Italy [Teta Srl] WWW
- Poland [Sunsite] WWW FTP
- Portugal [lerianet] WWW FTP
- Russia [DirectNet] WWW
- Russia [IZHCOM] WWW FTP
- Russia [Scientific Center/Chernogolovka] FTP
- Romania [Timisoara] WWW FTP

- Romania [Bucharest] WWW FTP
- Sweden [Sunet] WWW FTP
- Switzerland [Sunsite] WWW FTP
- UK [Omnipotent/UK] WWW FTP
- UK [PLiG/UK] WWW FTP
- UK [SunSITE] WWW FTP
- Ukraine [PACO] WWW FTP

North America:

- Canada [Tryc] WWW
- USA [Hurricane Electric/San Jose] WWW
- USA [Netcasting/West Coast] FTP
- USA [Circle Net/North Carolina] WWW
- USA [Gina net/Florida] WWW
- USA [pingzero/Los Angeles] WWW
- USA [Wisconsin University/Wisconsin] WWW FTP
- USA [DIGEX] FTP

South America:

- Chile [Vision] WWW

Asia:

- China [Freecode] WWW
- Korea [KREONet] WWW
- Japan [Soft Agency] WWW
- Japan [Nagoya Syouka University] WWW FTP
- Singapore [HJC] WWW FTP
- Taiwan [HT] WWW

Australia:

- Australia [AARNet/Queensland] WWW FTP
- Australia [Tas] WWW FTP
- Australia [Blue Planet/Melbourne] WWW
- Australia [ITworks Consulting/Victoria] WWW

Africa:

- South-Africa [Mweb/] WWW
- South-Africa [The Internet Solution/Johannesburg] FTP

4.2 Operating systems supported by MySQL

We use GNU Autoconf so it is possible to port **MySQL** to all modern systems with working Posix threads and a C++ compiler. (To compile only the client code, a C++ compiler is required but not threads.) We use and develop the software ourselves primarily on Sun Solaris (versions 2.5 & 2.6) and to a lesser extent on RedHat Linux 5.0.

MySQL has been reported to compile sucessfully on the following operating system/thread package combinations. Note that for many operating systems, the native thread support works only in the latest versions.

- AIX 4.x with native threads
- BSDI 2.x with the included MIT-pthreads package
- BSDI 3.0, 3.1 and 4.x with native threads
- DEC UNIX 4.x with native threads
- FreeBSD 2.x with the included MIT-pthreads package
- FreeBSD 3.x with native threads
- HP-UX 10.20 with the included MIT-pthreads package
- HP-UX 11.x with the native threads.
- Linux 2.0+ with LinuxThreads 0.7.1 or `glibc` 2.0.7
- NetBSD 1.3/1.4 Intel and NetBSD 1.3 Alpha (Requires GNU make)
- OpenBSD 2.x with the included MIT-pthreads package
- OS/2 Warp 3, FixPack 29 and OS/2 Warp 4, FixPack 4
- SGI Irix 6.x with native threads
- Solaris 2.5, 2.6 and 2.7 with native threads on SPARC and x86
- SunOS 4.x with the included MIT-pthreads package
- SCO OpenServer with a recent port of the FSU Pthreads package
- SCO UnixWare 7.0.1
- Tru64 Unix
- Win95, Win98 and NT (the newest version is currently available only for users with a **MySQL** license or **MySQL** email support). For those who wish to test before they buy, we have released **MySQL** 3.21.29 (an older version) as shareware.

4.3 Which MySQL version to use

The first decision to make is whether you want to use the latest development release or the last stable release:

- Normally, if you are beginning to use **MySQL** for the first time or trying to port it to some system for which there is no binary distribution, we recommend going with the development release (currently 3.22.x). This is because there are usually no really serious bugs in the development release, and you can easily test it on your machine with the `crash-me` and benchmark tests. See Chapter 11 [Benchmarks], page 265.

- Otherwise, if you are running an old system and want to upgrade, but don't want to take chances with 3.22, you should upgrade to 3.21.33. We have tried to fix only fatal bugs and make small, relatively safe changes to that version.

The second decision to make is whether you want to use a source distribution or a binary distribution:

- If you want to run **MySQL** on a platform for which a current binary distribution exists, use that. Generally, it will be easier to install than a source distribution.
- If you want to read (and/or modify) the C and C++ code that makes up **MySQL**, you should get a source distribution. The source code is always the ultimate manual. Source distributions also contain more tests and examples than binary distributions.

The **MySQL** naming scheme uses release numbers that consist of three numbers and a suffix. For example, a release name like `mysql-3.21.17-beta` is interpreted like this:

- The first number (`3`) describes the file format. All version 3 releases have the same file format. When a version 4 appears, every table will have to be converted to the new format (nice tools for this will be included, of course).
- The second number (`21`) is the release level. Normally there are two to choose from. One is the release/stable branch (currently `21`) and the other is the development branch (currently `22`) . Normally both are stable, but the development version may have quirks, missing documentation on new features or may fail to compile on some systems.
- The third number (`17`) is the version number within the release level. This is incremented for each new distribution. Usually you want the latest version for the release level you have choosen.
- The suffix (`beta`) indicates the stability level of the release. The possible suffixes are:
  - `alpha` indicates that the release contains some large section of new code that hasn't been 100% tested. Known bugs (usually there are none) should be documented in the News section. See Appendix D [News], page 400. There are also new commands and extensions in most alpha releases.
  - `beta` means that all new code has been tested. No major new features were added. There should be no known bugs.
  - `gamma` is a beta that has been around a while and seems to work fine. This is what many other companies call a release.
  - If there is no suffix, it means that the version has been run for a while at many different sites with no reports of bugs other than platform-specific bugs. This is what we call a stable release.

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

Note that all releases have been tested at least with:

An internal test suite
: This is part of a production system for a customer. It has many tables with hundreds of megabytes of data.

The **MySQL** benchmark suite
: This runs a range of common queries. It is also a test to see whether the latest batch of optimizations actually made the code faster. See Chapter 11 [Benchmarks], page 265.

The `crash-me` test
: This tries to determine what features the database supports and what its capabilities and limitations are. See Chapter 11 [Benchmarks], page 265.

Another test is that we use the newest **MySQL** version in our internal production environment, on at least one machine. We have more than 100 gigabytes of data to work with.

4.4 How and when updates are released

MySQL is evolving quite rapidly here at TcX and we want to share this with other **MySQL** users. We try to make a release when we have very useful features that others seem to have a need for.

We also try to help out users who request features that are easy to implement. We also take note of what our licensed users want to have and we especially take note of what our extended email supported customers want and try to help them out.

No one has to download a new release. The News section will tell you if the new release has something you really want. See Appendix D [News], page 400.

We use the following policy when updating **MySQL**:

- For each minor update, the last number in the version string is incremented. When there are major new features or minor incompatibilities with previous versions, the second number in the version string is incremented. When the file format changes, the first number is increased.
- Stable tested releases are meant to appear about 1-2 times a year, but if small bugs are found, a release with only bug-fixes will be released.
- Working releases are meant to appear about every 1-8 weeks.
- Binary distributions for some platforms will be made by us for major releases. Other people may make binary distributions for other systems but probably less frequently.
- We usually make patches available as soon as we have located and fixed small bugs.
- For non-critical but annoying bugs, we will make patches available if they are sent to us. Otherwise we will combine many of them into a larger patch.
- If there is, by any chance, a fatal bug in a release we will make a new release as soon as possible. We would like other companies to do this, too. :)

The current stable release is 3.22; We have already moved active development to 3.23. Bugs will still be fixed in the stable version. We don't believe in a complete freeze, as this also leaves out bug fixes and things that "must be done". "Somewhat frozen" means that we may add small things that "almost surely will not affect anything that's already working".

4.5 Installation layouts

This section describes the default layout of the directories created by installing binary and source distributions.

A binary distribution is installed by unpacking it at the installation location you choose (typically '`/usr/local/mysql`') and creates the following directories in that location:

Directory	Contents of directory
'`bin`'	Client programs and the `mysqld` server
'`data`'	Log files, databases
'`include`'	Include (header) files
'`lib`'	Libraries
'`scripts`'	`mysql_install_db`
'`share/mysql`'	Error message files
'`sql-bench`'	Benchmarks

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

Directory	Contents of directory
'`bin`'	Client programs and scripts
'`include/mysql`'	Include (header) files
'`info`'	Documentation in Info format
'`lib/mysql`'	Libraries
'`libexec`'	The `mysqld` server
'`share/mysql`'	Error message files
'`sql-bench`'	Benchmarks and `crash-me` test
'`var`'	Databases and log files.

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

- The `mysqld` server is installed in the '`libexec`' directory rather than in the '`bin`' directory.
- The data directory is '`var`' rather than '`data`'.
- `mysql_install_db` is installed in the '`/usr/local/bin`' directory rather than in '`/usr/local/mysql/scripts`'.
- The header file and library directories are '`include/mysql`' and '`lib/mysql`' rather than '`include`' and '`lib`'.

4.6 Installing a MySQL binary distribution

You need the following tools to install a **MySQL** binary distribution:

- GNU `gunzip` to uncompress the distribution.
- A reasonable `tar` to unpack the distribution. GNU `tar` is known to work.

An alternative installation method under Linux is to use RPM (RedHat Package Manager) distributions. See Section 4.6.1 [Linux-RPM], page 37.

If you run into problems, **PLEASE ALWAYS USE** `mysqlbug` when posting questions to `mysql@lists.mysql.com`. Even if the problem isn't a bug, `mysqlbug` gathers system information that will help others solve your problem. By not using `mysqlbug`, you lessen the likelihood of getting a solution to your problem! You will find `mysqlbug` in the '`bin`' directory after you unpack the distribution. See Section 2.3 [Bug reports], page 15.

The basic commands you must execute to install and use a **MySQL** binary distribution are:

```
shell> gunzip < mysql-VERSION-OS.tar.gz | tar xvf -
shell> ln -s mysql-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db
shell> bin/safe_mysqld &
```

You can add new users using the `bin/mysql_setpermission` script if you install the `DBI` and `Msql-Mysql-modules` Perl modules.

Here follows a more detailed description:

To install a binary distribution, follow the steps below, then proceed to Section 4.15 [Post-installation], page 72, for post-installation setup and testing:

1. Pick the directory under which you want to unpack the distribution, and move into it. In the example below, we unpack the distribution under '`/usr/local`' and create a directory '`/usr/local/mysql`' into which **MySQL** is installed. (The following instructions therefore assume you have permission to create files in '`/usr/local`'. If that directory is protected, you will need to perform the installation as `root`.)
2. Obtain a distribution file from one of the sites listed in Section 4.1 [Getting **MySQL**], page 29.

   MySQL binary distributions are provided as compressed `tar` archives and have names like '`mysql-VERSION-OS.tar.gz`', where `VERSION` is a number (e.g., `3.21.15`), and `OS` indicates the type of operating system for which the distribution is intended (e.g., `pc-linux-gnu-i586`).
3. Unpack the distribution and create the installation directory:

   ```
   shell> gunzip < mysql-VERSION-OS.tar.gz | tar xvf -
   shell> ln -s mysql-VERSION-OS mysql
   ```

   The first command creates a directory named '`mysql-VERSION-OS`'. The second command makes a symbolic link to that directory. This lets you refer more easily to the installation directory as '`/usr/local/mysql`'.
4. Change into the installation directory:

   ```
   shell> cd mysql
   ```

   You will find several files and subdirectories in the `mysql` directory. The most important for installation purposes are the '`bin`' and '`scripts`' subdirectories.

   '`bin`' This directory contains client programs and the server You should add the full pathname of this directory to your `PATH` environment variable so that your shell finds the **MySQL** programs properly.

   '`scripts`' This directory contains the `mysql_install_db` script used to initialize the server access permissions.

5. If you would like to use `mysqlaccess` and have the **MySQL** distribution in some non-standard place, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the '`bin/mysqlaccess`' script at approximately line 18. Search for a line that looks like this:

   ```
   $MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
   ```

   Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, you will get a `broken pipe` error when you run `mysqlaccess`.
6. Create the **MySQL** grant tables (necessary only if you haven't installed **MySQL** before):

   ```
   shell> scripts/mysql_install_db
   ```
7. If you want to install support for the Perl `DBI/DBD` interface, see Section 4.10 [Perl support], page 48.
8. If you would like **MySQL** to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself, and in Section 4.15.3 [Automatic start], page 78.

After everything has been unpacked and installed, you should initialize and test your distribution.

You can start the **MySQL** server with the following command:

```
shell> bin/safe_mysqld &
```

Note that **MySQL** versions older than 3.22.10 started the **MySQL** server when you run `mysql_install_db`. This is no longer true!

See Section 4.15 [Post-installation], page 72.

4.6.1 Linux RPM notes

The recommended way to install **MySQL** on Linux is by using an RPM file. The **MySQL** RPMs are currently being built on a RedHat 5.2 system but should work on other versions of Linux that support `rpm` and use `glibc`.

If you have problems with an RPM file, for example `Sorry, the host 'xxxx' could not be looked up`, see Section 4.6.3.1 [Binary notes-Linux], page 39.

The RPM files you may want to use are:

- `MySQL-VERSION.i386.rpm`

  The **MySQL** server. You will need this unless you only want to connect to another **MySQL** server running on another machine.
- `MySQL-client-VERSION.i386.rpm`

  The standard **MySQL** client programs. You probably always want to install this package.
- `MySQL-bench-VERSION.i386.rpm`

  Tests and benchmarks. Requires Perl and msql-mysql-modules RPMs.
- `MySQL-devel-VERSION.i386.rpm`

  Libraries and include files needed if you want to compile other **MySQL** clients, such as the Perl modules.

- `MySQL-VERSION.src.rpm`

  This contains the source code for all of the above packages. It can also be used to try to build RPMs for other architectures (for example, Alpha or SPARC).

To see all files in an RPM package:

```
shell> rpm -qpl MySQL-VERSION.i386.rpm
```

To perform a standard minimal installation, run this command:

```
shell> rpm -i MySQL-VERSION.i386.rpm MySQL-client-VERSION.i386.rpm
```

To install just the client package:

```
shell> rpm -i MySQL-client-VERSION.i386.rpm
```

The RPM places data in '`/var/lib/mysql`'. The RPM also creates the appropriate entries in '`/sbin/rc.d/`' to start the server automatically at boot time. (This means that if you have performed a previous installation, you may want to make a copy of your previously-installed **MySQL** startup file if you made any changes to it, so you don't lose your changes.)

After installing the RPM file(s), go to the binary install section and use the instructions there, starting from the step that creates the **MySQL** grant tables. See Section 4.6 [Installing binary], page 36.

4.6.2 Building client programs

If you compile **MySQL** clients that you've written yourself or that you obtain from a third party, they must be linked using the `-lmysqlclient` option on the link command. You may also need to specify a `-L` option to tell the linker where to find the library. For example, if the library is installed in '`/usr/local/mysql/lib`', use `-L/usr/local/mysql/lib -lmysqlclient` on the link command.

For clients that use **MySQL** header files, you may need to specify a `-I` option when you compile them (for example, `-I/usr/local/mysql/include`), so the compiler can find the header files.

4.6.3 System-specific issues

The following sections indicate some of the issues that have been observed to occur on particular systems when installing **MySQL** from a binary distribution.

4.6.3.1 Linux notes

MySQL needs at least Linux 2.0.

The binary release is linked with `-static`, which means you not normally need not worry about which version of the system libraries you have. You need not install LinuxThreads, either. A program linked with `-static` is slightly bigger than a dynamically-linked program but also slightly faster (3-5%). One problem however is that you can't use user definable functions (UDFs) with a statically-linked program. If you are going to write or use UDF functions (this is something only for C or C++ programmers) you must compile **MySQL** yourself, using dynamic linking.

If you are using a `libc`-based system (instead of a `glibc2` system), you will probably get some problems with hostname resolving and getpwnam() with the binary release. (This is because `glibc` unfortunately depends on some external libraries to resolve hostnames and getwpent() , even when compiled with `-static`). In this case you probably get the following error message when you run `mysql_install_db`:

```
Sorry, the host 'xxxx' could not be looked up
```

or the following error when you try to run mysqld with the `--user` option:

```
getpwnam: No such file or directory
```

You can solve this problem one of the following ways:

- Get a **MySQL** source distribution (an RPM or the `tar` distribution) and install this instead.
- Execute `mysql_install_db --force`; This will not execute the `resolveip` test in `mysql_install_db`. The downside is that you can't use host names in the grant tables; you must use IP numbers instead (except for `localhost`). If you are using an old **MySQL** release that doesn't support `--force` you have to remove the `resolveip` test in `mysql_install` with an editor.
- Start mysqld with `su` instead of using `--user`.

The Linux-Intel binary and RPM releases of **MySQL** are configured for the highest possible speed. We are always trying to use the fastest stable compiler available.

MySQL Perl support requires Perl 5.004_03 or newer.

4.6.3.2 HP-UX notes

The binary distribution of **MySQL** for HP-UX is distributed as an HP depot file and as a tar file. To use the depot file you must be running at least HP-UX 10.x to have access to HP's software depot tools.

The HP version of **MySQL** was compiled on an HP 9000/8xx server under HP-UX 10.20, and uses MIT-pthreads. It is known to work well under this configuration. **MySQL** 3.22.26 and newer can also be built with HP's native thread package.

Other configurations that may work:

- HP 9000/7xx running HP-UX 10.20+
- HP 9000/8xx running HP-UX 10.30

The following configurations almost definitely won't work:

- HP 9000/7xx or 8xx running HP-UX 10.x where x < 2
- HP 9000/7xx or 8xx running HP-UX 9.x

To install the distribution, use one of the commands below, where `/path/to/depot` is the full pathname of the depot file:

- To install everything, including the server, client and development tools:

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.full
```

- To install only the server:

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.server
```

- To install only the client package:

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.client
```

- To install only the development tools:

```
shell> /usr/sbin/swinstall -s /path/to/depot mysql.developer
```

The depot places binaries and libraries in '`/opt/mysql`' and data in '`/var/opt/mysql`'. The depot also creates the appropriate entries in '`/sbin/init.d`' and '`/sbin/rc2.d`' to start the server automatically at boot time. Obviously, this entails being `root` to install.

To install the HP-UX tar distribution, you must have a copy of gnu tar.

4.7 Installing a MySQL source distribution

You need the following tools to build and install **MySQL** from source:

- GNU `gunzip` to uncompress the distribution.
- A reasonable `tar` to unpack the distribution. GNU `tar` is known to work.
- A working ANSI C++ compiler. `gcc` >= 2.8.1, `egcs` >= 1.0.2, SGI C++ and SunPro C++ are some of the compilers that are known to work. `libg++` is not needed when using `gcc`. `gcc` 2.7.x has a bug that makes it impossible to compile some perfectly legal C++ files, such as '`sql/sql_base.cc`'. If you only have `gcc` 2.7.x, you must upgrade your `gcc` to be able to compile **MySQL**.
- A good `make` program. GNU `make` is always recommended and is sometimes required. If you have problems, we recommend trying GNU `make` 3.75 or newer.

If you run into problems, **PLEASE ALWAYS USE** `mysqlbug` when posting questions to `mysql@lists.mysql.com`. Even if the problem isn't a bug, `mysqlbug` gathers system information that will help others solve your problem. By not using `mysqlbug`, you lessen the likelihood of getting a solution to your problem! You will find `mysqlbug` in the '`scripts`' directory after you unpack the distribution. See Section 2.3 [Bug reports], page 15.

4.7.1 Quick installation overview

The basic commands you must execute to install a **MySQL** source distribution are (from an unpacked `tar` file):

```
shell> configure
shell> make
shell> make install
shell> scripts/mysql_install_db
shell> /usr/local/mysql/bin/safe_mysqld &
```

If you start from a source RPM, then do the following.

```
shell> rpm --rebuild MySQL-VERSION.src.rpm
```

This will make a binary RPM that you can install.

You can add new users using the `bin/mysql_setpermission` script if you install the `DBI` and `Msql-Mysql-modules` Perl modules.

Here follows a more detailed description:

To install a source distribution, follow the steps below, then proceed to Section 4.15 [Post-installation], page 72, for post-installation initialization and testing.

1. Pick the directory under which you want to unpack the distribution, and move into it.
2. Obtain a distribution file from one of the sites listed in Section 4.1 [Getting **MySQL**], page 29.

   MySQL source distributions are provided as compressed `tar` archives and have names like '`mysql-VERSION.tar.gz`', where `VERSION` is a number like 3.23.3-alpha.
3. Unpack the distribution into the current directory:

   ```
   shell> gunzip < mysql-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named '`mysql-VERSION`'.
4. Change into the top-level directory of the unpacked distribution:

   ```
   shell> cd mysql-VERSION
   ```
5. Configure the release and compile everything:

   ```
   shell> ./configure --prefix=/usr/local/mysql
   shell> make
   ```

   When you run `configure`, you might want to specify some options. Run `./configure --help` for a list of options. Section 4.7.3 [`configure` options], page 42, discusses some of the more useful options.

   If `configure` fails, and you are going to send mail to `mysql@lists.mysql.com` to ask for assistance, please include any lines from '`config.log`' that you think can help solve the problem. Also include the last couple of lines of output from `configure` if `configure` aborts. Post the bug report using the `mysqlbug` script. See Section 2.3 [Bug reports], page 15.

   If the compile fails, see Section 4.8 [Compilation problems], page 44, for help with a number of common problems.
6. Install everything:

   ```
   shell> make install
   ```

   You might need to run this command as `root`.
7. Create the **MySQL** grant tables (necessary only if you haven't installed **MySQL** before):

   ```
   shell> scripts/mysql_install_db
   ```

   Note that **MySQL** versions older than 3.22.10 started the **MySQL** server when you run `mysql_install_db`. This is no longer true!
8. If you want to install support for the Perl `DBI/DBD` interface, see Section 4.10 [Perl support], page 48.
9. If you would like **MySQL** to start automatically when you boot your machine, you can copy `support-files/mysql.server` to the location where your system has its startup files. More information can be found in the `support-files/mysql.server` script itself, and in Section 4.15.3 [Automatic start], page 78.

After everything has been installed, you should initialize and test your distribution.

You can start the **MySQL** server with the following command, where `BINDIR` is the directory in which `safe_mysqld` is installed ('`/usr/local/bin`' by default):

```
shell> BINDIR/safe_mysqld &
```

If that command fails immediately with `mysqld daemon ended` then you can find some information in the file '`mysql-data-directory/'hostname'.err`'. The likely reason is that you already have another `mysqld` server running. See Section 19.3 [Multiple servers], page 322.

See Section 4.15 [Post-installation], page 72.

4.7.2 Applying patches

Sometimes patches appear on the mailing list or are placed in the patches area of the **MySQL** FTP site.

To apply a patch from the mailing list, save the message in which the patch appears in a file, change into the top-level directory of your **MySQL** source tree and run these commands:

```
shell> patch -p1 < patch-file-name
shell> rm config.cache
shell> make clean
```

Patches from the FTP site are distributed as plain text files or as files compressed with `gzip` files. Apply a plain patch as shown above for mailing list patches. To apply a compressed patch, change into the top-level directory of your **MySQL** source tree and run these commands:

```
shell> gunzip < patch-file-name.gz | patch -p1
shell> rm config.cache
shell> make clean
```

After applying a patch, follow the instructions for a normal source install, beginning with the `./configure` step. After running the `make install` step, restart your **MySQL** server.

You may need to bring down any currently running server before you run `make install`. (Use `mysqladmin shutdown` to do this.) Some systems do not allow you to install a new version of a program if it replaces the version that is currently executing.

4.7.3 Typical `configure` options

The `configure` script gives you a great deal of control over how you configure your **MySQL** distribution. Typically you do this using options on the `configure` command line. You can also affect `configure` using certain environment variables. For a list of options supported by `configure`, run this command:

```
shell> ./configure --help
```

Some of the more commonly-used `configure` options are described below:

- To compile just the **MySQL** client libraries and client programs and not the server, use the `--without-server` option:

  ```
  shell> ./configure --without-server
  ```

  If you don't have a C++ compiler, `mysql` will not compile (it is the one client program that requires C++). In this case, you can remove the code in `configure` that tests for the C++ compiler and then run `./configure` with the `--without-server` option. The compile step will still try to build `mysql`, but you can ignore any warnings about

'`mysql.cc`'. (If `make` stops, try `make -k` to tell it to continue with the rest of the build even if errors occur.)

- If you don't want your log files and database directories located under '`/usr/local/var`', use a `configure` command something like one of these:

```
shell> ./configure --prefix=/usr/local/mysql
shell> ./configure --prefix=/usr/local \
           --localstatedir=/usr/local/mysql/data
```

  The first command changes the installation prefix so that everything is installed under '`/usr/local/mysql`' rather than the default of '`/usr/local`'. The second command preserves the default installation prefix, but overrides the default location for database directories (normally '`/usr/local/var`') and changes it to `/usr/local/mysql/data`.

- If you are using Unix and you want the **MySQL** socket located somewhere other than the default location (normally in the directory '`/tmp`' or '`/var/run`', use a `configure` command like this:

```
shell> ./configure --with-unix-socket-path=/usr/local/mysql/tmp/mysql.sock
```

  Note that the given file must be an absolute pathname!

- If you want to compile statically-linked programs (e.g., to make a binary distribution, to get more speed or to work around problems with some RedHat distributions), run `configure` like this:

```
shell> ./configure --with-client-ldflags=-all-static \
           --with-mysqld-ldflags=-all-static
```

- If you are using `gcc` and don't have `libg++` or `libstdc++` installed, you can tell `configure` to use `gcc` as your C++ compiler:

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

  When you use `gcc` as your C++ compiler, it will not attempt to link in `libg++` or `libstdc++`.

  If the build fails and produces errors about your compiler or linker not being able to create the shared library '`libmysqlclient.so.#`' ('`#`' is a version number), you can work around this problem by giving the `--disable-shared` option to `configure`. In this case, `configure` will not build a shared `libmysqlclient.so.#` library.

- You can configure **MySQL** not to use `DEFAULT` column values for non-`NULL` columns (i.e., columns that are not allowed to be `NULL`). This causes `INSERT` statements to generate an error unless you explicitly specify values for all columns that require a non-`NULL` value. To suppress use of default values, run `configure` like this:

```
shell> CXXFLAGS=-DDONT_USE_DEFAULT_FIELDS ./configure
```

- By default, **MySQL** uses the ISO-8859-1 (Latin1) character set. To change the default set, use the `--with-charset` option:

```
shell> ./configure --with-charset=CHARSET
```

  `CHARSET` may be one of `big5`, `cp1251`, `cp1257`, `czech`, `danish`,`dec8`, `dos`, `euc_kr`, `german1`, `hebrew`, `hp8`, `hungarian`, `koi8_ru`, `koi8_ukr`, `latin1`, `latin2`, `sjis`, `swe7`, `tis620`, `ujis`, `usa7`, `win1251` or `win1251ukr`. See Section 9.1.1 [Character sets], page 240.

Note that if you want to change the character set, you must do a `make distclean` between configurations!

If you want to convert characters between the server and the client, you should take a look at the `SET OPTION CHARACTER SET` command. See Section 7.24 [SET OPTION], page 199.

Warning: If you change character sets after having created any tables, you will have to run `isamchk -r -q` on every table. Your indexes may be sorted incorrectly otherwise. (This can happen if you install **MySQL**, create some tables, then reconfigure **MySQL** to use a different character set and reinstall it.)

- To configure **MySQL** with debugging code, use the `--with-debug` option:

```
shell> ./configure --with-debug
```

  This causes a safe memory allocator to be included that can find some errors and that provides output about what is happening. See Section G.1 [Debugging server], page 444.

- Options that pertain to particular systems can be found in the system-specific sections later in this chapter. See Section 4.11 [Source install system issues], page 51.

4.8 Problems compiling?

All **MySQL** programs compile cleanly for us with no warnings on Solaris using `gcc`. On other systems, warnings may occur due to differences in system include files. See Section 4.9 [MIT-pthreads], page 46, for warnings that may occur when using MIT-pthreads. For other problems, check the list below.

The solution to many problems involves reconfiguring. If you do need to reconfigure, take note of the following:

- If `configure` is run after it already has been run, it may use information that was gathered during its previous invocation. This information is stored in '`config.cache`'. When `configure` starts up, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `configure`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first, since they were compiled using different configuration options.

To prevent old configuration information or object files from being used, run these commands before rerunning `configure`:

```
shell> rm config.cache
shell> make clean
```

Alternatively, you can run `make distclean`.

The list below describes some of the problems compiling **MySQL** that have been found to occur most often:

- If you get errors when compiling '`sql_yacc.cc`' such as the ones shown below, you have probably run out of memory or swap space:

```
Internal compiler error: program cc1plus got fatal signal 11
  or
Out of virtual memory
  or
Virtual memory exhausted
```

The problem is that `gcc` requires huge amounts of memory to compile '`sql_yacc.cc`' with inline functions. Try running `configure` with the `--with-low-memory` option:

```
shell> ./configure --with-low-memory
```

This option causes `-fno-inline` to be added to the compile line if you are using `gcc` and `-O0` if you are using something else. You should try the `--with-low-memory` option even if you have so much memory and swap space that you think you can't possibly have run out. This problem has been observed to occur even on systems with generous hardware configurations, and the `--with-low-memory` option usually fixes it.

- By default, `configure` picks `c++` as the compiler name and GNU `c++` links with `-lg++`. If you are using `gcc`, that behavior can cause problems during configuration such as this:

  ```
  configure: error: installation or configuration problem:
  C++ compiler cannot create executables.
  ```

  You might also observe problems during compilation related to `g++`, `libg++` or `libstdc++`.

  One cause of these problems is that you may not have `g++`, or you may have `g++` but not `libg++` or `libstdc++`. Take a look at the '`config.log`' file. It should contain the exact reason why your `c++` compiler didn't work! To work around these problems, you can use `gcc` as your C++ compiler. Try setting the environment variable `CXX` to `"gcc -O3"`. For example:

  ```
  shell> CXX="gcc -O3" ./configure
  ```

  This works because `gcc` compiles C++ sources as well as `g++` does, but does not link in `libg++` or `libstdc++` by default.

  Another way to fix these problems, of course, is to install `g++`, `libg++` and `libstdc++`.

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

  ```
  making all in mit-pthreads
  make: Fatal error in reader: Makefile, line 18:
  Badly formed macro assignment
    or
  make: file 'Makefile' line 18: Must be a separator (:
    or
  pthread.h: No such file or directory
  ```

  Solaris and FreeBSD are known to have troublesome `make` programs.

  GNU `make` version 3.75 is known to work.

- If you want to define flags to be used by your C or C++ compilers, do so by adding the flags to the `CFLAGS` and `CXXFLAGS` environment variables. You can also specify the compiler names this way using `CC` and `CXX`. For example:

```
shell> CC=gcc
shell> CFLAGS=-O6
shell> CXX=gcc
shell> CXXFLAGS=-O6
shell> export CC CFLAGS CXX CXXFLAGS
```

See Section 4.14 [TcX binaries], page 71, for a list of flag definitions that have been found to be useful on various systems.

- If you get an error message like this, you need to upgrade your `gcc` compiler:

  ```
  client/libmysql.c:273: parse error before `__attribute__'
  ```

  `gcc` 2.8.1 is known to work, but we recommend using `egcs` 1.0.3a or newer instead.

- If you get errors such as those shown below when compiling `mysqld`, `configure` didn't correctly detect the type of the last argument to `accept()`, `getsockname()` or `getpeername()`:

  ```
  cxx: Error: mysqld.cc, line 645: In this statement, the referenced
       type of the pointer value "&length" is "unsigned long", which
       is not compatible with "int".
  new_sock = accept(sock, (struct sockaddr *)&cAddr, &length);
  ```

  To fix this, edit the 'config.h' file (which is generated by `configure`). Look for these lines:

  ```
  /* Define as the base type of the last arg to accept */
  #define SOCKET_SIZE_TYPE XXX
  ```

  Change `XXX` to `size_t` or `int`, depending on your operating system. (Note that you will have to do this each time you run `configure`, since `configure` regenerates 'config.h'.)

- The 'sql_yacc.cc' file is generated from 'sql_yacc.yy'. Normally the build process doesn't need to create 'sql_yacc.cc', because **MySQL** comes with an already-generated copy. However, if you do need to recreate it, you might encounter this error:

  ```
  "sql_yacc.yy", line xxx fatal: default action causes potential...
  ```

  This is a sign that your version of `yacc` is deficient. You probably need to install `bison` (the GNU version of `yacc`) and use that instead.

- If you need to debug `mysqld` or a **MySQL** client, run `configure` with the `--with-debug` option, then recompile and link your clients with the new client library. See Section G.2 [Debugging client], page 446.

4.9 MIT-pthreads notes

This section describes some of the issues involved in using MIT-pthreads.

Note that on Linux you should NOT use MIT-pthreads but install LinuxThreads! See Section 4.11.5 [Linux], page 54.

If your system does not provide native thread support, you will need to build **MySQL** using the MIT-pthreads package. This includes most FreeBSD systems, SunOS 4.x, Solaris 2.4 and earlier, and some others. See Section 4.2 [Which OS], page 32.

- On most systems, you can force MIT-pthreads to be used by running `configure` with the `--with-mit-threads` option:

```
shell> ./configure --with-mit-threads
```

Building in a non-source directory is not supported when using MIT-pthreads, because we want to minimize our changes to this code.

- MIT-pthreads doesn't support the `AF_UNIX` protocol used to implement Unix sockets. This means that if you compile using MIT-pthreads, all connections must be made using TCP/IP (which is a little slower). If you find after building **MySQL** that you cannot connect to the local server, it may be that your client is attempting to connect to `localhost` using a Unix socket as the default. Try making a TCP/IP connection with `mysql` by using a host option (`-h` or `--host`) to specify the local host name explicitly.
- The checks that determine whether or not to use MIT-pthreads occur only during the part of the configuration process that deals with the server code. If you have configured the distribution using `--without-server` to build only the client code, clients will not know whether or not MIT-pthreads is being used and will use Unix socket connections by default. Since Unix sockets do not work under MIT-pthreads, this means you will need to use `-h` or `--host` when you run client programs.
- When **MySQL** is compiled using MIT-pthreads, system locking is disabled by default for performance reasons. You can tell the server to use system locking with the `--use-locking` option.
- Sometimes the pthread `bind()` command fails to bind to a socket without any error message (at least on Solaris). The result is that all connections to the server fail. For example:

  ```
  shell> mysqladmin version
  mysqladmin: connect to server at '' failed;
  error: 'Can't connect to mysql server on localhost (146)'
  ```

  The solution to this is to kill the `mysqld` server and restart it. This has only happened to us when we have forced the server down and done a restart immediately.
- With MIT-pthreads, the `sleep()` system call isn't interruptible with `SIGINT` (break). This is only noticeable when you run `mysqladmin --sleep`. You must wait for the `sleep()` call to terminate before the interrupt is served and the process stops.
- When linking you may receive warning messages like these (at least on Solaris); they can be ignored:

  ```
  ld: warning: symbol `_iob' has differing sizes:
      (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
  file /usr/lib/libc.so value=0x140);
      /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
  ld: warning: symbol `__iob' has differing sizes:
      (file /my/local/pthreads/lib/libpthread.a(findfp.o) value=0x4;
  file /usr/lib/libc.so value=0x140);
      /my/local/pthreads/lib/libpthread.a(findfp.o) definition taken
  ```

- Some other warnings also can be ignored:

  ```
  implicit declaration of function `int strtoll(...)'
  implicit declaration of function `int strtoul(...)'
  ```

- We haven't gotten `readline` to work with MIT-pthreads. (This isn't needed, but may be interesting for someone.)

4.10 Perl installation comments

4.10.1 Installing Perl on Unix

Perl support for **MySQL** is provided by means of the `DBI/DBD` client interface. See Section 20.5 [Perl], page 367. The Perl `DBD/DBI` client code requires Perl 5.004 or later. The interface *will not work* if you have an older version of Perl.

MySQL Perl support also requires that you've installed **MySQL** client programming support. If you installed **MySQL** from RPM files, client programs are in the client RPM, but client programming support is in the developer RPM. Make sure you've installed the latter RPM.

As of release 3.22.8, Perl support is distributed separately from the main **MySQL** distribution. If you want to install Perl support, the files you will need can be obtained from `http://www.mysql.com/Contrib`.

The Perl distributions are provided as compressed `tar` archives and have names like '`MODULE-VERSION.tar.gz`', where `MODULE` is the module name and `VERSION` is the version number. You should get the `Data-Dumper`, `DBI`, and `Msql-Mysql-modules` distributions and install them in that order. The installation procedure is shown below. The example shown is for the `Data-Dumper` module, but the procedure is the same for all three distributions.

1. Unpack the distribution into the current directory:

   ```
   shell> gunzip < Data-Dumper-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named '`Data-Dumper-VERSION`'.

2. Change into the top-level directory of the unpacked distribution:

   ```
   shell> cd Data-Dumper-VERSION
   ```

3. Build the distribution and compile everything:

   ```
   shell> perl Makefile.PL
   shell> make
   shell> make test
   shell> make install
   ```

The `make test` command is important, because it verifies that the module is working. Note that when you run that command during the `Msql-Mysql-modules` installation to exercise the interface code, the **MySQL** server must be running or the test will fail.

It is a good idea to rebuild and reinstall the `Msql-Mysql-modules` distribution whenever you install a new release of **MySQL**, particularly if you notice symptoms such as all your `DBI` scripts dumping core after you upgrade **MySQL**.

If you don't have the right to install Perl modules in the system directory or if you to install local Perl modules, the following reference may help you:

`http://www.iserver.com/support/contrib/perl5/modules.html`

Look under the heading `Installing New Modules that Require Locally Installed Modules`.

4.10.2 Installing ActiveState Perl on Win32

To install the **MySQL** `DBD` module with ActiveState Perl on Win32, you should do the following:

- Open a DOS shell.
- If required, set the HTTP_proxy variable. For example, you might try: `set HTTP_proxy=my.proxy.com:3128`
- Start the PPM program: `C:\perl\bin\ppm.pl`
- If you have not already done so, install `DBI`: `install DBI`
- If this succeeds, install `DBD::mysql`: `http://www.mysql.com/Contrib/ppd/DBD-mysql.ppd`

If you can't get the above to work, you should instead install the **MyODBC** driver and connect to **MySQL** server through ODBC.

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn","$user","$password") ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

4.10.3 Installing the MySQL Perl distribution on Win32

The **MySQL** Perl distribution contains `DBI`, `DBD:MySQL` and `DBD:ODBC`.

- Get the Perl distribution for Win32 from http://www.mysql.com/download.html.
- Unzip the distribution in `C:` so that you get a '`C:\PERL`' directory.
- Add the directory '`C:\PERL\BIN`' to your path.
- Add the directory '`C:\PERL\BIN\MSWin32-x86-thread`' or '`C:\PERL\BIN\MSWin32-x86`' to your path.
- Test that `perl` works by executing `perl -v` in a DOS shell.

4.10.4 Problems using the Perl `DBI/DBD` interface

If Perl reports that it can't find the `../mysql/mysql.so` module, then the problem is probably that Perl can't locate the shared library '`libmysqlclient.so`'.

You can fix this by any of the following methods:

- Compile the `Msql-Mysql-modules` distribution with `perl Makefile.PL -static` rather than `perl Makefile.PL`
- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably '`/usr/lib`' or '`/lib`').
- On `Linux` you can add the pathname of the directory where `libmysqlclient.so` is located to the '`/etc/ld.so.conf`' file.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable.

If you get the following errors from `DBD-mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the '`mysql.so`' library gets built (check the output from `make` for '`mysql.so`' when you compile the Perl client). The `-L` option should specify the pathname of the directory where '`libgcc.a`' is located on your system.

Another cause of this problem may be that Perl and **MySQL** aren't both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

If you want to use the Perl module on a system that doesn't support dynamic linking (like SCO) you can generate a static version of Perl that includes `DBI` and `DBD-mysql`. The way this works is that you generate a version of Perl with the `DBI` code linked in and install it on top of your current Perl. Then you use that to build a version of Perl that additionally has the `DBD` code linked in, and install that.

On SCO, you must have the following environment variables set:

```
shell> LD_LIBRARY_PATH=/lib:/usr/lib:/usr/local/lib:/usr/progressive/lib
or
shell> LD_LIBRARY_PATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:/usr/progressive/l
shell> LIBPATH=/usr/lib:/lib:/usr/local/lib:/usr/ccs/lib:/usr/progressive/lib:/usr/
shell> MANPATH=scohelp:/usr/man:/usr/local1/man:/usr/local/man:/usr/skunk/man:
```

First, create a Perl that includes a statically-linked `DBI` by running these commands in the directory where your `DBI` distribution is located:

```
shell> perl Makefile.PL LINKTYPE=static
shell> make
shell> make install
shell> make perl
```

Then you must install the new Perl. The output of `make perl` will indicate the exact `make` command you will need to execute to perform the installation. On SCO, this is `make -f Makefile.aperl inst_perl MAP_TARGET=perl`.

Next, use the just-created Perl to create another Perl that also includes a statically-linked `DBD::mysql` by running these commands in the directory where your `Msql-Mysql-modules` distribution is located:

```
shell> perl Makefile.PL LINKTYPE=static
shell> make
shell> make install
shell> make perl
```

Finally, you should install this new Perl. Again, the output of `make perl` indicates the command to use.

4.11 System-specific issues

The following sections indicate some of the issues that have been observed to occur on particular systems when installing **MySQL** from a source distribution.

4.11.1 Solaris notes

On Solaris, you may run into trouble even before you get the **MySQL** distribution unpacked! Solaris `tar` can't handle long file names, so you may see an error like this when you unpack **MySQL**:

```
x mysql-3.22.12-beta/bench/Results/ATIS-mysql_odbc-NT_4.0-cmp-db2,informix,ms-sql,m
tar: directory checksum error
```

In this case, you must use GNU `tar` (`gtar`) to unpack the distribution. You can find a precompiled copy for Solaris at http://www.mysql.com/Downloads/.

Sun native threads work only on Solaris 2.5 and higher. For 2.4 and earlier versions, **MySQL** will automatically use MIT-pthreads. See Section 4.9 [MIT-pthreads], page 46.

If you get the following error from configure:

```
checking for restartable system calls... configure: error can not run test
programs while cross compiling
```

This means that you have something wrong with your compiler installation! In this case you should upgrade your compiler to a newer version. You may also be able to solve this problem by inserting the following row into the `config.cache` file:

```
ac_cv_sys_restartable_syscalls=${ac_cv_sys_restartable_syscalls='no'}
```

If you are using Solaris on a SPARC, the recommended compiler is `egcs` 1.1.2 or newer. You can find this at http://egcs.cygnus.com/. Note that `egs` 1.1.1 and `gcc` 2.8.1 don't work reliably on SPARC!

The recommended `configure` line when using `egcs` 1.1.2 is:

```
shell> CC=gcc CFLAGS="-O6" \
       CXX=gcc CXXFLAGS="-O6 -felide-constructors -fno-exceptions -fno-rtti" \
       ./configure --prefix=/usr/local/mysql --with-low-memory
```

If you have the Sun Workshop 4.2 compiler, you can run `configure` like this:

CC=cc CFLAGS="-xstrconst -Xa -xO4 -native -mt" CXX=CC CXXFLAGS="-xO4 -native -noex -mt" ./configure –prefix=/usr/local/mysql

```
shell> CC=cc CFLAGS="-Xa -fast -xO4 -native -xstrconst -mt" \
       CXX=CC CXXFLAGS="-noex -XO4 -mt" \
       ./configure
```

You may also have to edit the `configure` script to change this line:

```
#if !defined(__STDC__) || __STDC__ != 1
```

to this:

```
#if !defined(__STDC__)
```

If you turn on `__STDC__` with the `-Xc` option, the Sun compiler can't compile with the Solaris '`pthread.h`' header file. This is a Sun bug (broken compiler or broken include file).

If `mysqld` issues the error message shown below when you run it, you have tried to compile **MySQL** with the Sun compiler without enabling the multi-thread option (`-mt`):

```
libc internal error: _rmutex_unlock: rmutex not held
```

Add `-mt` to `CFLAGS` and `CXXFLAGS` and try again.

If you get the following error when compiling **MySQL** with `gcc`, it means that your `gcc` is not configured for your version of Solaris!

```
shell> gcc -O3 -g -O2 -DDBUG_OFF  -o thr_alarm ...
./thr_alarm.c: In function ‘signal_hand’:
./thr_alarm.c:556: too many arguments to function ‘sigwait’
```

The proper thing to do in this case is to get the newest version of `egcs` and compile it with your current `gcc` compiler! At least for Solaris 2.5, almost all binary versions of `gcc` have old, unusable include files that will break all programs that use threads (and possibly other programs)!

Solaris doesn't provide static versions of all system libraries (`libpthreads` and `libdl`), so you can't compile **MySQL** with `--static`. If you try to do so, you will get the error:

```
ld: fatal: library -ldl: not found
```

If too many processes try to connect very rapidly to `mysqld`, you will see this error in the **MySQL** log:

```
Error in accept: Protocol error
```

You might try starting the server with the `--set-variable back_log=50` option as a workaround for this.

If you are linking your own **MySQL** client, you might get the following error when you try to execute it:

```
ld.so.1: ./my: fatal: libmysqlclient.so.#: open failed: No such file or directory
```

The problem can be avoided by one of the following methods:

- Link the client with the following flag (instead of `-Lpath`): `-Wl,r/full-path-to-libmysqlclient.so`.
- Copy `libmysqclient.so` to '`/usr/lib`'.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

4.11.2 Solaris 2.7 notes

You can normally use a Solaris 2.6 binary on Solaris 2.7. Most of the Solaris 2.6 issues also apply for Solaris 2.7.

Solaris 2.7 has some bugs in the include files. You may see the following error when you use `gcc`:

```
/usr/include/widec.h:42: warning: ‘getwc’ redefined
/usr/include/wchar.h:326: warning: this is the location of the previous
definition
```

If this occurs, you can do the following to fix the problem:

Copy `/usr/include/widec.h` to `.../lib/gcc-lib/os/gcc-version/include` and change line 41 from:

```
#if     !defined(lint) && !defined(__lint)

to

#if     !defined(lint) && !defined(__lint) && !defined(getwc)
```

Alternatively, you can edit '`/usr/include/widec.h`' directly. Either way, after you make the fix, you should remove '`config.cache`' and run `configure` again!

If you get errors like this when you run `make`, it's because `configure` didn't detect the '`curses.h`' file (probably because of the error in `/usr/include/widec.h`:

```
In file included from mysql.cc:50:
/usr/include/term.h:1060: syntax error before `,'
/usr/include/term.h:1081: syntax error before `;'
```

The solution to this is to do one of the following steps:

- Edit '`/usr/include/widec.h`' as indicted above and rerun configure
- Remove the `#define HAVE_TERM` line from '`config.h`' file and run `make` again.
- Configure with `CFLAGS=-DHAVE_CURSES CXXFLAGS=-DCURSES ./configure`

4.11.3 Solaris x86 notes

If you are using `gcc` or `egcs` on Solaris x86 and you experience problems with core dumps under load, you should use the following `configure` command:

```
shell> CC=gcc CFLAGS="-O6 -fomit-frame-pointer" \
       CXX=gcc \
       CXXFLAGS="-O6 -fomit-frame-pointer -felide-constructors -fno-exceptions -fno
       ./configure --prefix=/usr/local/mysql
```

This will avoid problems with the `libstdc++` library and with C++ exceptions.

If this doesn't help, you should compile a debug version and run it with a trace file or under `gdb`. See Section G.1 [Debugging server], page 444.

4.11.4 SunOS 4 notes

On SunOS 4, MIT-pthreads is needed to compile **MySQL**, which in turn means you will need GNU `make`.

Some SunOS 4 systems have problems with dynamic libraries and `libtool`. You can use the following `configure` line to avoid this problem:

```
shell> ./configure --disable-shared --with-mysqld-ldflags=-all-static
```

When compiling `readline`, you may get warnings about duplicate defines. These may be ignored.

When compiling `mysqld`, there will be some `implicit declaration of function` warnings. These may be ignored.

4.11.5 Linux notes (all Linux versions)

MySQL uses LinuxThreads on Linux. If you are using an old Linux version that doesn't have `glibc2`, you must install LinuxThreads before trying to compile **MySQL**. http://www.mysql.com/Downloads/Linux

If you can't start `mysqld` or if `mysql_install_db` doesn't work, please continue reading! This only happens on Linux system with problems in the LinuxThreads or `libc`/`glibc` libraries. There are a lot of simple workarounds to get **MySQL** to work! The simplest is

to use the binary version of **MySQL** (not the RPM) for Linux x86. One nice aspect of this version is that it's probably 10% faster than any version you would compile yourself! See Section 10.3 [Compile and link options], page 249.

One known problem with the binary distribution is that with older Linux systems that use `libc` (like RedHat 4.x or Slackware), you will get some non-fatal problems with hostname resolution See Section 4.6.3.1 [Binary notes-Linux], page 39.

`isamchk` hangs with `libc.so.5.3.12`. Upgrading to the newest `libc` fixes this problem.

When using LinuxThreads you will see a minimum of three processes running. These are in fact threads. There will be one thread for the LinuxThreads manager, one thread to handle connections, and one thread to handle alarms and signals.

If you see a dead `mysqld` daemon process with `ps`, this usually means that you have found a bug in **MySQL** or you have got a corrupted table. See Section 18.1 [Crashing], page 307.

If you are using LinuxThreads and `mysqladmin shutdown` doesn't work, you must upgrade to LinuxThreads 0.7.1 or newer.

If you are using RedHat, you might get errors like this:

```
/usr/bin/perl is needed...
/usr/sh is needed...
/usr/sh is needed...
```

If so, you should upgrade your version of `rpm` to '`rpm-2.4.11-1.i386.rpm`' and '`rpm-devel-2.4.11-1.i386.rpm`' (or later).

You can get the upgrades of libraries to RedHat 4.2 from `ftp://ftp.redhat.com/updates/4.2/i386`. Or `http://www.sunsite.unc.edu/pub/Linux/distributions/redhat/code/rpm/` for other distributions.

If you are linking your own **MySQL** client and get the error:

```
ld.so.1: ./my: fatal: libmysqlclient.so.4: open failed: No such file or directory
```

when executing them, the problem can be avoided by one of the following methods:

- Link the client with the following flag (instead of `-Lpath`): `-Wl,r/path-libmysqlclient.so`.
- Copy `libmysqclient.so` to '`/usr/lib`'.
- Add the pathname of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable before running your client.

If you are using the Fujitsu compiler (`fcc / FCC`) you will have some problems compiling **MySQL** because the Linux header files are very `gcc` oriented.

The following `configure` line should work with `fcc/FCC`:

```
CC=fcc CFLAGS="-O -K fast -K lib -K omitfp -Kpreex -D_GNU_SOURCE -DCONST=const -DNO
```

4.11.5.1 Linux-x86 notes

MySQL requires `libc` version 5.4.12 or newer. It's known to work with `libc` 5.4.46. `glibc` version 2.0.6 and later should also work. There have been some problems with the `glibc` RPMs from RedHat so if you have problems, check whether or not there are any updates! The `glibc` 2.0.7-19 and 2.0.7-29 RPMs are known to work.

On some older Linux distributions, `configure` may produce an error like this:

```
Syntax error in sched.h. Change _P to __P in the /usr/include/sched.h file.
See the Installation chapter in the Reference Manual.
```

Just do what the error message says and add an extra underscore to the `_P` macro that has only one underscore, then try again.

You may get some warnings when compiling; those shown below can be ignored:

```
mysqld.cc -o objs-thread/mysqld.o
mysqld.cc: In function `void init_signals()':
mysqld.cc:315: warning: assignment of negative value `-1' to `long unsigned int'
mysqld.cc: In function `void * signal_hand(void *)':
mysqld.cc:346: warning: assignment of negative value `-1' to `long unsigned int'
```

In Debian GNU/Linux, if you want **MySQL** to start automatically when the system boots, do the following:

```
shell> cp support-files/mysql.server /etc/init.d/mysql.server
shell> /usr/sbin/update-rc.d mysql.server defaults 99
```

`mysql.server` can be found in the '`share/mysql`' directory under the **MySQL** installation directory, or in the '`support-files`' directory of the **MySQL** source tree.

If `mysqld` always core dumps when it starts up, the problem may be that you have an old '`/lib/libc.a`'. Try renaming it, then remove '`sql/mysqld`' and do a new `make install` and try again. This problem has been reported on some Slackware installations. RedHat 5.0 has also a similar problem with some new `glibc` versions. See Section 4.11.5.2 [Linux-RedHat50], page 55.

If you get the following error when linking `mysqld`, it means that your '`libg++.a`' is not installed correctly:

```
/usr/lib/libc.a(putc.o): In function `_IO_putc':
putc.o(.text+0x0): multiple definition of `_IO_putc'
```

You can avoid using '`libg++.a`' by running `configure` like this:

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

4.11.5.2 RedHat 5.0 notes

If you have any problems with **MySQL** on RedHat, you should start by upgrading `glibc` to the newest possible version!

If you install all the official RedHat patches (including `glibc-2.0.7-19` and `glibc-devel-2.0.7-19`), both the binary and source distributions of **MySQL** should work without any trouble!

The updates are needed since there is a bug in `glibc` 2.0.5 in how `pthread_key_create` variables are freed. With `glibc` 2.0.5, you must use a statically-linked **MySQL** binary distribution. If you want to compile from source, you must install the corrected version of LinuxThreads from `http://www.mysql.com/Downloads/Linux` or upgrade your `glibc`.

If you have an incorrect version of `glibc` or LinuxThreads, the symptom is that `mysqld` crashes after each connection. For example, `mysqladmin version` will crash `mysqld` when it finishes!

Another symptom of incorrect libraries is that `mysqld` crashes at once when it starts. On some Linux systems, this can be fixed by configuring like this:

```
shell> ./configure --with-mysqld-ldflags=-all-static
```

On Redhat 5.0, the easy way out is to install the `glibc` 2.0.7-19 RPM and run `configure` **without** the `--with-mysqld-ldflags=-all-static` option.

For the source distribution of `glibc` 2.0.7, a patch that is easy to apply and is tested with **MySQL** may be found at:

`http://www.mysql.com/Download/Linux/glibc-2.0.7-total-patch.tar.gz`

If you experience crashes like these when you build **MySQL**, you can always download the newest binary version of **MySQL**. This is statically-linked to avoid library conflicts and should work on all Linux systems!

MySQL comes with an internal debugger that can generate trace files with a lot of information that can be used to find and solve a wide range of different problems. See Section G.1 [Debugging server], page 444.

4.11.5.3 RedHat 5.1 notes

The `glibc` of RedHat 5.1 (`glibc` 2.0.7-13) has a memory leak, so to get a stable **MySQL** version, you must upgrade `glibc` to 2.0.7-19, downgrade `glibc` or use a binary version of `mysqld`. If you don't do this, you will encounter memory problems (out of memory, etc., etc.). The most common error in this case is:

```
Can't create a new thread (errno 11). If you are not out of available
memory, you can consult the manual for any possible OS dependent bug
```

After you have upgraded to `glibc` 2.0.7-19, you can configure **MySQL** with dynamic linking (the default), but you **cannot** run `configure` with the `--with-mysqld-ldflags=-all-static` option until you have installed `glibc` 2.0.7-19 from source!

You can check which version of `glibc` you have with `rpm -q glibc`.

4.11.5.4 Linux-SPARC notes

In some implementations, `readdir_r()` is broken. The symptom is that `SHOW DATABASES` always returns an empty set. This can be fixed by removing `HAVE_READDIR_R` from '`config.h`' after configuring and before compiling.

Some problems will require patching your Linux installation. The patch can be found at `http://www.mysql.com/patches/Linux-sparc-2.0.30.diff`. This patch is against the Linux distribution '`sparclinux-2.0.30.tar.gz`' that is available at `vger.rutgers.edu` (a version of Linux that was never merged with the official 2.0.30). You must also install LinuxThreads 0.6 or newer.

Thanks to `jacques@solucorp.qc.ca` for this information.

4.11.5.5 Linux-Alpha notes

The big problem on Linux-Alpha is that there are still some problems with threads in `glibc` on this platform. You should start by getting the newest `glibc` version you can find.

Note that before you run any programs that use threads (like `mysqld`, `thr_alarm` or `thr_lock`), you should raise the shared memory limit (with `ulimit`). The **MySQL** benchmarks are known to fail if you forget to do this!

Configure **MySQL** with the following command:

```
shell> CC=gcc CCFLAGS="-Dalpha_linux_port" \
       CXX=gcc CXXFLAGS="-O3 -Dalpha_linux_port -felide-constructors -fno-exceptions
       ./configure --prefix=/usr/local/mysql
```

Try to compile `mysys/thr_lock` and `mysys/thr_alarm`. Test that these programs work! (Invoke each one with no arguments. Each should end with `test_succeeded` if everything was okay.)

After installing **MySQL**, uncomment the `ulimit` command in `safe_mysqld` and add options to increase shared memory.

Note that Linux-Alpha is still an alpha-quality platform for **MySQL**. With the newest `glibc`, you have a very good chance of it working.

If you have problems with signals (**MySQL** dies unexpectedly under high load) you may have found an OS bug with threads and signals. In this case you can tell **MySQL** not to use signals by configuring with:

```
shell> CFLAGS=-DDONT_USE_THR_ALARM \
       CXXFLAGS=-DDONT_USE_THR_ALARM \
       ./configure ...
```

This doesn't affect the performance of **MySQL**, but has the side effect that you can't kill clients that are "sleeping" on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client will die when it issues its next command.

4.11.5.6 MkLinux notes

MySQL should work on MkLinux with the newest `glibc` package (tested with `glibc` 2.0.7).

4.11.5.7 Qube2 Linux notes

To get **MySQL** to work on Qube2, (Linux Mips), you need the newest `glibc` libraries (`glibc-2.0.7-29C2` is known to work). You must also use the `egcs` C++ compiler (`egcs-1.0.2-9` or newer).

4.11.6 Alpha-DEC-Unix notes

When compiling threaded programs under Digital UNIX, the documentation recommends using the `-pthread` option for `cc` and `cxx` and the libraries `-lmach -lexc` (in addition to `-lpthread`). You should run `configure` something like this:

```
shell> CC="cc -pthread" CXX="cxx -pthread -O" \
       ./configure --with-named-thread-libs="-lpthread -lmach -lexc -lc"
```

When compiling `mysqld`, you may see a couple of warnings like this:

```
mysqld.cc: In function void handle_connections()':
mysqld.cc:626: passing long unsigned int *' as argument 3 of
accept(int,sockadddr *, int *)'
```

You can safely ignore these warnings. They occur because `configure` can detect only errors, not warnings.

If you start the server directly from the command line, you may have problems with it dying when you log out. (When you log out, your outstanding processes receive a `SIGHUP` signal.) If so, try starting the server like this:

```
shell> nohup mysqld [options] &
```

`nohup` causes the command following it to ignore any `SIGHUP` signal sent from the terminal. Alternatively, start the server by running `safe_mysqld`, which invokes `mysqld` using `nohup` for you.

4.11.7 Alpha-DEC-OSF1 notes

If you have problems compiling and have DEC `CC` and `gcc` installed, try running `configure` like this:

```
shell> CC=cc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 \
       ./configure --prefix=/usr/local/mysql
```

If you get problems with the '`c_asm.h`' file, you can create and use a 'dummy' '`c_asm.h`' file with:

```
shell> touch include/c_asm.h
shell> CC=gcc CFLAGS=-I./include \
       CXX=gcc CXXFLAGS=-O3 \
       ./configure --prefix=/usr/local/mysql
```

On OSF1 V4.0D and compiler "DEC C V5.6-071 on Digital UNIX V4.0 (Rev. 878)" the compiler had some strange behavior (undefined `asm` symbols). `/bin/ld` also appears to be broken (problems with `_exit undefined` errors occuring while linking `mysqld`). On this system, we have managed to compile **MySQL** with the following `configure` line, after replacing `/bin/ld` with the version from OSF 4.0C:

```
shell> CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

In some versions of OSF1, the `alloca()` function is broken. Fix this by removing the line in '`config.h`' that defines '`HAVE_ALLOCA`'.

The `alloca()` function also may have an incorrect prototype in `/usr/include/alloca.h`. This warning resulting from this can be ignored.

`configure` will use the following thread libraries automatically: `--with-named-thread-libs="-lpthread -lmach -lexc -lc"`.

When using `gcc`, you can also try running `configure` like this:

```
shell> CFLAGS=-D_PTHREAD_USE_D4 CXX=gcc CXXFLAGS=-O3 ./configure ....
```

If you have problems with signals (**MySQL** dies unexpectedly under high load) you may have found an OS bug with threads and signals. In this case you can tell **MySQL** not to use signals by configuring with:

```
shell> CFLAGS=-DDONT_USE_THR_ALARM \
       CXXFLAGS=-DDONT_USE_THR_ALARM \
       ./configure ...
```

This doesn't affect the performance of **MySQL**, but has the side effect that you can't kill clients that are "sleeping" on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client will die when it issues its next command.

4.11.8 SGI-Irix notes

You may have to undefine some things in '`config.h`' after running `configure` and before compiling.

In some Irix implementations, the `alloca()` function is broken. If the `mysqld` server dies on some `SELECT` statements, remove the lines from '`config.h`' that define `HAVE_ALLOC` and `HAVE_ALLOCA_H`. If `mysqladmin create` doesn't work, remove the line from '`config.h`' that defines `HAVE_READDIR_R`. You may have to remove the `HAVE_TERM_H` line as well.

SGI recommends that you install all of the patches on this page as a set: http://support.sgi.com/surfzone/patches/patchset/6.2_indigo.rps.html

At the very minimum, you should install the latest kernel rollup, the latest `rld` rollup, and the latest `libc` rollup.

You definately need all the POSIX patches on this page, for pthreads support:

http://support.sgi.com/surfzone/patches/patchset/6.2_posix.rps.html

If you get the something like the following error when compiling '`mysql.cc`':

```
"/usr/include/curses.h", line 82: error(1084): invalid combination of type
```

Then type the following in the top-level directory of your **MySQL** source tree:

```
shell> extra/replace bool curses_bool < /usr/include/curses.h > include/curses.h
shell> make
```

There have also been reports of scheduling problems. If only one thread is running, things go slow. Avoid this by starting another client. This may lead to a 2-to-10-fold increase in execution speed thereafter for the other thread. This is a poorly-understood problem with Irix threads; you may have to improvise to find solutions until this can be fixed.

If you are compiling with `gcc`, you can use the following `configure` command:

```
shell> CC=gcc CXX=gcc CXXFLAGS=-O3 \
       ./configure --prefix=/usr/local/mysql --with-thread-safe-client --with-named
```

4.11.9 FreeBSD notes

The easiest and therefor the preferred way to install is to use the mysql-server and mysql-client ports available on http://www.freebsd.org

Using these gives you:

- A working **MySQL** with all optimisations known to work on your version of FreeBSD enabled.
- Automatic configuration and build.
- Startup scripts installed in /usr/local/etc/rc.d
- Ability to see which files that are installed with pkg_info -L. And to remove them all with pkg_delete if you no longer want **MySQL** on that machine.

It is recomended to use MIT-pthreads on FreeBSD 2.x and native threads on versions 3 and up. It is possible to run with with native threads on some late 2.2.x versions but you may encounter problems shutting down mysqld.

Be sure to have your name resolver setup correct. Otherwise you may experience resolver delays or failures when connecting to mysqld.

Make sure that the `localhost` entry in the '`/etc/hosts`' file is correct (otherwise you will have problems connecting to the database). The '`/etc/hosts`' file should start with a line:

```
127.0.0.1       localhost localhost.your.domain
```

If you notice that `configure` will use MIT-pthreads, you should read the MIT-pthreads notes. See Section 4.9 [MIT-pthreads], page 46.

If you get an error from `make install` that it can't find '`/usr/include/pthreads`', `configure` didn't detect that you need MIT-pthreads. This is fixed by executing these commands:

```
shell> rm config.cache
shell> ./configure --with-mit-threads
```

The behavior of FreeBSD `make` is slightly different from that of GNU `make`. If you have `make`-related problems, you should install GNU `make`.

FreeBSD is also known to have a very low default file handle limit. See Section 18.11 [Not enough file handles], page 315. Uncomment the ulimit -n section in safe_mysqld or raise the limits for the mysqld user in /etc/login.conf (and rebuild it witg cap_mkdb /etc/login.conf) also be sure you set the appropriate Class for this user in the password file if you are not using the default (use: chpass mysqld-user-name)

If you have a problem with `SELECT NOW()` returning values in GMT and not your local time, you have to set the `TZ` environment variable to your current timezone. This should be done for the environment in which the server runs, for example, in `safe_mysqld` or `mysql.server`.

To get a secure and stable system you should only use FreeBSD kernels that are marked `-STABLE`

4.11.10 NetBSD notes

To compile on NetBSD you need GNU `make`. Otherwise the compile will crash when `make` tries to run `lint` on C++ files.

4.11.11 BSD/OS notes

4.11.11.1 BSD/OS 2.x notes

If you get the following error when compiling **MySQL**, your `ulimit` value for virtual memory is too low:

```
item_func.h: In method `Item_func_ge::Item_func_ge(const Item_func_ge &)':
item_func.h:28: virtual memory exhausted
make[2]: *** [item_func.o] Error 1
```

Try using `ulimit -v 80000` and run `make` again. If this doesn't work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

If you are using `gcc`, you may also use have to use the `--with-low-memory` flag for `configure` to be able to compile '`sql_yacc.cc`'.

If you have a problem with `SELECT NOW()` returning values in GMT and not your local time, you have to set the `TZ` environment variable to your current timezone. This should be done for the environment in which the server runs, for example in `safe_mysqld` or `mysql.server`.

4.11.11.2 BSD/OS 3.x notes

Upgrade to BSD/OS 3.1. If that is not possible, install BSDIpatch M300-038.

Use the following command when configuring **MySQL**:

```
shell> env CXX=shlicc++ CC=shlicc2 \
       ./configure \
           --prefix=/usr/local/mysql \
           --localstatedir=/var/mysql \
           --without-perl \
           --with-unix-socket-path=/var/mysql/mysql.sock
```

The following is also known to work:

```
shell> env CC=gcc CXX=gcc CXXFLAGS=-O3 \
       ./configure \
           --prefix=/usr/local/mysql \
           --with-unix-socket-path=/var/mysql/mysql.sock
```

You can change the directory locations if you wish, or just use the defaults by not specifying any locations.

If you have problems with performance under heavy load, try using the `--skip-thread-priority` option to `safe_mysqld`! This will run all threads with the same priority; on BSDI 3.1, this gives better performance (at least until BSDI fixes their thread scheduler).

If you get the error `virtual memory exhausted` while compiling, you should try using `ulimit -v 80000` and run `make` again. If this doesn't work and you are using `bash`, try switching to `csh` or `sh`; some BSDI users have reported problems with `bash` and `ulimit`.

4.11.11.3 BSD/OS 4.x notes

BSDI 4.x has some thread related bugs. If you want to use **MySQL** on this, you should install all thread related patches. At least M400-023 should be installed.

4.11.12 SCO notes

The current port is tested only on a "sco3.2v5.0.4" and "sco3.2v5.0.5" system. There has also been a lot of progress on a port to "sco 3.2v4.2".

1. For OpenServer 5.0.X you need to use GDS in Skunkware 95 (95q4c). This is necessary because GNU `gcc` 2.7.2 in Skunkware 97 does not have GNU `as`. You can also use `egcs` 1.1.2 or newer http://www.egcs.com/. If you are using `egcs` 1.1.2 you have to execute the following command:

   ```
   shell> cp -p /usr/include/pthread/stdtypes.h /usr/local/lib/gcc-lib/i386-pc-sco3
   ```

2. You need the port of GCC 2.5.? for this product and the Development system. They are required on this version of SCO UNIX. You cannot just use the GCC Dev system.
3. You should get the FSU Pthreads package and install it first. This can be found at http://www.cs.wustl.edu/~schmidt/ACE_wrappers/FSU-threads.tar.gz. You can also get a precompiled package from ftp://www.mysql.com/pub/mysql/Downloads/SCO/FSU-thre
4. FSU Pthreads can be compiled with SCO UNIX 4.2 with tcpip. Or OpenServer 3.0 or Open Desktop 3.0 (OS 3.0 ODT 3.0), with the SCO Development System installed using a good port of GCC 2.5.X ODT or OS 3.0 you will need a good port of GCC 2.5.? There are a lot of problems without a good port. The port for this product requires the SCO UNIX Development system. Without it, you are missing the libraries and the linker that is needed.
5. To build FSU Pthreads on your system, do the following:
   1. Run `./configure` in the '`threads/src`' directory and select the SCO OpenServer option. This command copies '`Makefile.SCO5`' to '`Makefile`'.
   2. Run `make`.
   3. To install in the default '`/usr/include`' directory, login as root, then `cd` to the '`thread/src`' directory, and run `make install`.
6. Remember to use GNU `make` when making **MySQL**.
7. On OSR 5.0.5, you should use the following configure line:

   ```
   shell> CC="gcc -DSCO" CXX="gcc -DSCO" ./configure
   ```

   The `-DSCO` is needed to help configure detect some thread functions properly. If you forget `-DSCO`, you will get the following error message while compiling:

   ```
   my_pthread.c: In function 'my_pthread_mutex_init':
   my_pthread.c:374: 'pthread_mutexattr_default' undeclared (first use this functio
   ```
8. If you don't start `safe_mysqld` as root, you probably will get only the default 110 open files per process. `mysqld` will write a note about this in the log file.
9. With SCO 3.2V5.0.5, you should use a FSU Pthreads version 3.5c or newer. The following `configure` command should work:

   ```
   shell> CC="gcc -belf" ./configure --prefix=/usr/local/mysql --disable-shared
   ```
10. With SCO 3.2V4.2, you should use a FSU Pthreads version 3.5c or newer. The following `configure` command should work:

    ```
    shell> CFLAGS="-D_XOPEN_XPG4" CXX=gcc CXXFLAGS="-D_XOPEN_XPG4" \
           ./configure \
               --with-debug --prefix=/usr/local/mysql \
               --with-named-thread-libs="-lgthreads -lsocket -lgen -lgthreads" \
               --with-named-curses-libs="-lcurses"
    ```

    You may get some problems with some include files. In this case, you can find new SCO-specific include files at ftp://www.mysql.com/pub/mysql/Downloads/SCO/SCO-3.2v4.2-includes.t You should unpack this file in the '`include`' directory of your **MySQL** source tree.

SCO development notes:

- **MySQL** should automatically detect FSU Pthreads and link `mysqld` with `-lgthreads -lsocket -lgthreads`.

- The SCO development libraries are reentrant in FSU Pthreads. SCO claims that its libraries' functions are reentrant, so they must be reentrant with FSU Pthreads. FSU Pthreads on OpenServer tries to use the SCO scheme to make reentrant library.
- FSU Pthreads (at least the version at `www.mysql.com`) comes linked with GNU `malloc`. If you encounter problems with memory usage, make sure that '`gmalloc.o`' is included in '`libgthreads.a`' and '`libgthreads.so`'.
- In FSU Pthreads, the following system calls are pthreads-aware: `read()`, `write()`, `getmsg()`, `connect()`, `accept()`, `select()` and `wait()`.

If you want to install DBI on SCO, you have to edit the '`Makefiles`' in DBI-xxx and each subdirectory:

```
OLD:                                    NEW:
CC = cc                                 CC = gcc -belf
CCCDLFLAGS = -KPIC -W1,-Bexport         CCCDLFLAGS = -fpic
CCDLFLAGS = -wl,-Bexport                CCDLFLAGS =

LD = ld                                 LD = gcc -belf -G -fpic
LDDLFLAGS = -G -L/usr/local/lib         LDDLFLAGS = -L/usr/local/lib
LDFLAGS = -belf -L/usr/local/lib        LDFLAGS = -L/usr/local/lib

LD = ld                                 LD = gcc -belf -G -fpic
OPTIMISE = -Od                  OPTIMISE = -O1

OLD:
CCCFLAGS = -belf -dy -w0 -U M_XENIX -DPERL_SCO5 -I/usr/local/include

NEW:
CCFLAGS = -U M_XENIX -DPERL_SCO5 -I/usr/local/include
```

This is because the Perl dynaloader will not load the `DBI` modules if they were compiled with `icc` or `cc`.

Perl works best when compiled with `cc`.

4.11.13 SCO Unixware 7.0 notes

You must use a version of **MySQL** at least as recent as 3.22.13, since that version fixes some portability problems under Unixware.

We have been able to compile **MySQL** with the following `configure` command on UnixWare 7.0.1:

```
shell> CC=cc CXX=CC ./configure --prefix=/usr/local/mysql
```

4.11.14 IBM-AIX notes

Automatic detection of `xlC` is missing from Autoconf, so a `configure` command something like this is needed when using the IBM compiler:

```
shell> CC="xlc_r -ma -O3 -qstrict -DHAVE_INT_8_16_32" \
       CXX="xlC_r -ma -O3 -qstrict -DHAVE_INT_8_16_32" \
```

```
./configure
```

If you are using `egcs` to compile **MySQL**, you **MUST** use the `-fno-exceptions` flag, as the exception handling in `egcs` is not thread-safe! (This is tested with `egcs` 1.1.) We recommend the following `configure` line with `egcs` and `gcc` on AIX:

```
shell> CXX=gcc \
       CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti" \
       ./configure --prefix=/home/monty --with-debug --with-low-memory
```

If you have problems with signals (**MySQL** dies unexpectedly under high load) you may have found an OS bug with threads and signals. In this case you can tell **MySQL** not to use signals by configuring with:

```
shell> CFLAGS=-DDONT_USE_THR_ALARM CXX=gcc \
       CXXFLAGS="-felide-constructors -fno-exceptions -fno-rtti -DDONT_USE_THR_ALARM" \
       ./configure --prefix=/home/monty --with-debug --with-low-memory
```

This doesn't affect the performance of **MySQL**, but has the side effect that you can't kill clients that are "sleeping" on a connection with `mysqladmin kill` or `mysqladmin shutdown`. Instead, the client will die when it issues its next command.

4.11.15 HP-UX notes

There are a couple of "small" problems when compiling **MySQL** on HP-UX. We recommend that you use `gcc` instead of the HP-UX native compiler, because `gcc` produces better code!

We recommend one to use gcc 2.95 on HP-UX. Don't use high optimization flags (like -O6) as this may not be safe on HP-UX.

Note that MIT-pthreads can't be compiled with the HP-UX compiler, because it can't compile `.S` (assembler) files.

The following configure line should work:

```
CFLAGS="-DHPUX -I/opt/dce/include" CXXFLAGS="-DHPUX -I/opt/dce/include -felide-cons
```

If you are compiling `gcc` 2.95 yourself, you should NOT link it with the DCE libraries (`libdce.a` or `libcma.a`) if you want to compile **MySQL** with MIT-pthreads. If you mix the DCE and MIT-pthreads packages you will get a `mysqld` to which you cannot connect. Remove the DCE libraries while you compile `gcc` 2.95!

4.12 Win32 notes

This section describes installation and use of **MySQL** on Win32. This is also described in the '`README`' file that comes with the **MySQL** Win32 distribution.

4.12.1 Installing MySQL on Win32

If you don't have a registered version of **MySQL**, you should first download the shareware version from:

MySQL 3.21.29

If you plan to connect to **MySQL** from some other program, you will probably also need the **MyODBC** driver. You can find this at the **MySQL** download page.

To install either distribution, unzip it in some empty directory and run the `Setup.exe` program.

Installation takes place in '`C:\mysql`'.

4.12.2 Starting MySQL on Win95 / Win98

MySQL uses TCP/IP to connect a client to a server. (This will allow any machine on your network to connect to your **MySQL** server). Because of this, you must install TCP/IP on your machine before starting **MySQL**. You can find TCP/IP on your Windows CD-ROM.

Note that if you are using an old Win95 release (for example OSR2), it's likely that you have an old Winsock package! **MySQL** requires Winsock 2! You can get the newest Winsock from Microsoft. Win98 has as default the new Winsock 2 library, so the above doesn't apply for Win98.

There are 2 different **MySQL** servers you can use:

`mysqld`	Compiled with full debugging and automatic memory allocation checking
`mysqld-opt`	Optimized for a Pentium processor.

Both of the above should work on any Intel processor >= i386.

To start the `mysqld` server, you should start a MS-DOS window and type:

```
C:\mysql\bin\mysqld
```

This will start `mysqld` in the background without a window.

You can kill the **MySQL** server by executing:

```
C:\mysql\bin\mysqladmin -u root shutdown
```

Note that Win95/Win98 don't support creation of named pipes. On Win95/Win98, you can only use named pipes to connect to a remote **MySQL** running on an NT server.

4.12.3 Starting MySQL on NT

The Win95/Win98 section also applies to **MySQL** on NT, with the following differences:

To get **MySQL** to work with TCP/IP, you must install service pack 3 (or newer)!

For NT, the server name is `mysqld-nt`. Normally you should install **MySQL** as a service on NT:

```
C:\mysql\bin\mysqld-nt --install
```

(You could use the `mysqld` or `mysqld-opt` servers on NT, but those cannot be started as a service or use named pipes.)

You can start and stop the **MySQL** service with:

```
NET START mysql
NET STOP mysql
```

Note that in this case you can't use any other options for `mysqld-nt`!

You can also run `mysqld-nt` as a standalone program on NT if you need to start `mysqld-nt` with any options! If you start `mysqld-nt` without options on NT, `mysqld-nt` tries to starts itself as a service with the default service options. If you have stopped `mysqld-nt`, you have to start it with `NET START mysql`.

The service is installed with the name `MySql`. Once installed, it must be started using Services Control Manager (SCM) Utility (found in Control Panel) or by using the `NET START MySQL` command. If any options are desired, they must be specified as "Startup parameters" in the SCM utility before you start the **MySQL** service. Once running, `mysqld-nt` can be stopped using `mysqladmin` or from the SCM utility or by using the command `NET STOP MySQL`. If you use SCM to stop `mysqld-nt`, there is a strange message from SCM about `mysqld shutdown normally`. When run as a service, `mysqld-nt` has no access to a console and so no messages can be seen.

On NT you can get the following service error messages:

Permission Denied	Means that it cannot find `mysqld-nt.exe`
Cannot Register	Means that the path is incorrect

If you have problems installing `mysqld-nt` as a service, try starting it with the full path:

```
C:\mysql\bin\mysqld --install
```

If this doesn't work, you can get `mysqld-nt` to start properly by fixing the path in the registry!

If you don't want to start `mysqld-nt` as a service, you can start it as follows:

```
C:\mysql\bin\mysqld-nt --standalone
```

or

```
C:\mysql\bin\mysqld-nt --standalone --debug
```

The last version gives you a debug trace in '`C:\mysqld.trace`'.

4.12.4 Running MySQL on Win32

MySQL supports TCP/IP on all Win32 platforms and named pipes on NT. The default is to use named pipes for local connections on NT and TCP/IP for all other cases if the client has TCP/IP installed. The host name specifies which protocol is used:

Host name	**protocol**
NULL (none)	On NT, try named pipes first; if that doesn't work, use TCP/IP. On Win95/Win98, TCP/IP is used.
.	Named pipes
localhost	TCP/IP to current host
hostname	TCP/IP

You can force a **MySQL** client to use named pipes by specifying the `--pipe` option. Use the `--socket` option to specify the name of the pipe.

You can test whether or not **MySQL** is working by executing the following commands:

```
C:\mysql\bin\mysqlshow
C:\mysql\bin\mysqlshow -u root mysql
C:\mysql\bin\mysqladmin version status proc
C:\mysql\bin\mysql test
```

By default, **MySQL**-Win32 is configured to be installed in '`C:\mysql`'. If you want to install **MySQL** elsewhere, install it in '`C:\mysql`', then move the installation to where you want it. If you do move **MySQL**, you must tell `mysqld` where everything is by supplying options to `mysqld`. Use `C:\mysql\bin\mysqld --help` to display all options! If, for example, you

have moved the **MySQL** distribution to '`D:\programs\mysql`', you must start `mysqld` with: `D:\programs\mysql\bin\mysqld --basedir D:\programs\mysql`

With the registered version of **MySQL**, you can also create a '`C:\my.cnf`' file that holds any default options for the **MySQL** server. Copy the file '`\mysql\my-example.cnf`' to '`C:\my.cnf`' and edit this to suit your setup. Note that you should specify all paths with `/` instead of `\`. If you use `\`, you need to specify this twice, as `\` is the escape character in **MySQL**. See Section 4.15.4 [Option files], page 79.

If `mysqld` is slow to answer to connections on Win95/Win98, there is probably a problem with your DNS. In this case, start `mysqld` with `--skip-name-resolve` and use only `localhost` and IP numbers in the **MySQL** grant tables. You can also avoid DNS when connecting to a `mysqld-nt` **MySQL** server running on NT by using the `--pipe` argument to specify use of named pipes. This works for most **MySQL** clients.

There are two versions of the **MySQL** command line tool:

`mysql` Compiled on native Win32, which offers very limited text editing capabilities.

`mysqlc` Compiled with the Cygnus GNU compiler and libraries, which offers `readline` editing.

If you want to use `mysqlc.exe`, you must copy '`C:\mysql\lib\cygwinb19.dll`' to '`\windows\system`' (or similar place).

The default privileges on Win32 give all local users full privileges to all databases. To make **MySQL** more secure, you should set a password for all users and remove the row in the `mysql.user` table that has `Host='localhost'` and `User=''`.

You should also add a password for the `root` user:

```
C:\mysql\bin\mysql mysql
mysql> DELETE FROM user WHERE Host='localhost' AND User='';
mysql> QUIT
C:\mysql\bin\mysqladmin reload
C:\mysql\bin\mysqladmin -u root password your_password
```

After you've set the password, if you want to take down the `mysqld` server, you can do so using this command:

```
mysqladmin --user=root --password=your_password shutdown
```

4.12.5 Connecting to a remote MySQL from Win32 with SSH

Here is a note about how to connect to get a secure connection to remote MySQL server with SSH (by David Carlson).

- Install SSH client on your windows machine - I used a free SSH client from http://www.doc.ic.ac.uk/~ci2/ssh/. Other useful links: http://www.npaci.edu/Security/npaci_security_software.html and http://www.npaci.edu/Security/samples/ssh32_windows/index.html.
- Start SSH. Set Host Name = yourmysqlserver name or IP address. Set userid=your userid to log in to your server
- Click on "local forwards". Set `local port: 3306`, `host: localhost`, `remote port: 3306`

- Save everything, otherwise you'll have to redo it the next time.
- Log in to your server with SSH.
- Start some ODBC application (for example Access)
- Create a new file and link to mySQL using the ODBC driver the same way you normally do except for server, user "localhost".

That's it. It works very well with a direct Internet connection. I'm having problems with SSH conflicting with my Win95 network and Wingate - but that'll be the topic of a posting on another software company's usegroup!

4.12.6 MySQL-Win32 compared to Unix MySQL

MySQL-Win32 has by now proven itself to be very stable. This version of **MySQL** has the same features as the corresponding Unix version with the following exceptions:

Win95 and threads
: Win95 leaks about 200 bytes of main memory for each thread creation. Because of this, you shouldn't run `mysqld` for an extended time on Win95 if you do many connections, since each connection in **MySQL** creates a new thread! WinNT and Win98 don't suffer from this bug.

Blocking read
: **MySQL** uses a blocking read for each connection. This means that:

  - A connection will not be disconnected automatically after 8 hours, as happens with the Unix version of **MySQL**.
  - If a connection "hangs," it's impossible to break it without killing **MySQL**.
  - `mysqladmin kill` will not work on a sleeping connection.
  - `mysqladmin shutdown` can't abort as long as there are sleeping connections.

  We plan to fix this in the near future.

UDF functions
: For the moment, **MySQL**-Win32 does not support user definable functions.

`DROP DATABASE`
: You can't drop a database that is in use by some thread.

Killing MySQL from the task manager
: You can't kill **MySQL** from the task manager or with the shutdown utility in Windows95. You must take it down with `mysqladmin shutdown`.

Case-insensitive names
: Filenames are case insensitive on Win32, so database and table names are also case insensitive in **MySQL** for Win32. The only restriction is that database and table names must be given in the same case throughout a given statement. The following query would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

The '\' directory character

Pathname components in Win95 are separated by '\' characters, which is also the escape character in **MySQL**. If you are using `LOAD DATA INFILE` or `SELECT ... INTO OUTFILE`, you must double the '\' character or use Unix style filenames '`/`' characters:

```
LOAD DATA INFILE "C:\\tmp\\skr.txt" INTO TABLE skr;
SELECT * FROM skr INTO OUTFILE 'C:/tmp/skr.txt';
```

`Can't open named pipe` error

If you use the shareware version of **MySQL**-Win32 on NT with the newest mysql-clients you will get the following error:

```
error 2017: can't open named pipe to host: . pipe...
```

This is because the release version of **MySQL** uses named pipes on NT by default. You can avoid this error by using the `--host=localhost` option to the new **MySQL** clients or create a file '`C:\my.cnf`' that contains the following information:

```
[client]
host = localhost
```

`Access denied for user` error

If you get the error `Access denied for user: 'some-user@unknown' to database 'mysql'` when accessing a **MySQL** server on the same machine, this means that **MySQL** can't resolve your host name properly.

To fix this, you should create a file '`\windows\hosts`' with the following information:

```
127.0.0.1       localhost
```

Here are some open issues for anyone who might want to help us with the Win32 release:

- Make a single user `MYSQL.DLL` server. This should include everything in a standard **MySQL** server, except thread creation. This will make **MySQL** much easier to use in applications that don't need a true client/server and don't need to access the server from other hosts.
- Add some nice "start" and "shutdown" icons to the **MySQL** installation.
- Create a tool to manage registry entries for the **MySQL** startup options. The registry entry reading is already coded into `mysqld.cc`, but it should be recoded to be more "parameter" oriented. The tool should also be able to update the '`\my.cnf`' file if the user would prefer to use this instead of the registry.
- When registering `mysqld` as a service with `--install` (on NT) it would be nice if you could also add default options on the command line. For the moment, the workaround is to update the '`C:\my.cnf`' file instead.
- When you suspend a laptop running Win95, the `mysqld` daemon doesn't accept new connections when the laptop is resumed. We don't know if this is a problem with Win95, TCP/IP or **MySQL**.
- It would be real nice to be able to kill `mysqld` from the task manager. For the moment, you must use `mysqladmin shutdown`.

- Port `readline` to Win32 for use in the `mysql` command line tool.
- GUI versions of the standard **MySQL** clients (`mysql`, `mysqlshow`, `mysqladmin`, and `mysqldump`) would be nice.
- It would be nice if the socket "read" and "write" functions in '`net.c`' were interruptible. This would make it possible to kill open threads with `mysqladmin kill` on Win32.
- Documentation of which Windows programs work with **MySQL**-Win32/**MyODBC** and what must be done to get them working.
- `mysqld` always starts in the `"C"` locale and not in the default locale. We would like to have `mysqld` use the current locale for the sort order.
- Port `sqlclient` to Win32 (almost done) and add more features to it!
- Add more options to MysqlManager.
- Change the communication protocol between the server and client to use Windows internal communication instead of sockets and TCP/IP.
- Implement UDF functions with `.DLL`s.
- Add macros to use the faster thread-safe increment/decrement methods provided by Win32.

Other Win32-specific issues are described in the '`README`' file that comes with the **MySQL**-Win32 distribution.

4.13 OS/2 notes

MySQL uses quite a few open files. Because of this, you should add something like the following to your '`CONFIG.SYS`' file:

```
SET EMXOPT=-c -n -h1024
```

If you don't do this, you will probably run into the following error:

```
File 'xxxx' not found (Errcode: 24)
```

When using **MySQL** with OS/2 Warp 3, FixPack 29 or above is required. With OS/2 Warp 4, FixPack 4 or above is required. This is a requirement of the Pthreads library. **MySQL** must be installed in a partition that supports long file names such as HPFS, FAT32, etc.

The '`INSTALL.CMD`' script must be run from OS/2's own '`CMD.EXE`' and may not work with replacement shells such as '`4OS2.EXE`'.

The '`scripts/mysql-install-db`' script has been renamed: it is now called '`install.cmd`' and is a REXX script which will set up the default **MySQL** security settings and create the WorkPlace Shell icons for **MySQL**.

Dynamic module support is compiled in but not fully tested. Dynamic modules should be compiled using the Pthreads runtime library.

```
gcc -Zdll -Zmt -Zcrtdll=pthrdrtl -I../include -I../regex -I.. \
    -o example udf_example.cc -L../lib -lmysqlclient udf_example.def
mv example.dll example.udf
```

Note: Due to limitations in OS/2, UDF module name stems must not exceed 8 characters. Modules are stored in the '`/mysql2/udf`' directory; the `safe-mysqld.cmd` script will put this directory in the `BEGINLIBPATH` environment variable. When using UDF modules, specified

extensions are ignored — it is assumed to be '`.udf`'. For example, in Unix, the shared module might be named '`example.so`' and you would load a function from it like this:

```
CREATE FUNCTION metaphon RETURNS STRING SONAME "example.so";
```

Is OS/2, the module would be named '`example.udf`', but you would not specify the module extension:

```
CREATE FUNCTION metaphon RETURNS STRING SONAME "example";
```

4.14 TcX binaries

As a service, TcX provides a set of binary distributions of **MySQL** that are compiled at TcX or at sites where customers kindly have given us access to their machines.

These distributions are generated with `scripts/make_binary_distribution` and are configured with the following compilers and options:

SunOS 4.1.4 2 sun4c with `gcc` 2.7.2.1

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--disable-shared
```

SunOS 5.5.1 sun4u with `egcs` 1.0.3a

```
CC=gcc CFLAGS="-O6 -fomit-frame-pointer" CXX=gcc CXXFLAGS="-O6
-fomit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql --with-low-memory
```

SunOS 5.6 sun4u with `egcs` 2.90.27

```
CC=gcc CFLAGS="-O6 -fomit-frame-pointer" CXX=gcc CXXFLAGS="-O6
-fomit-frame-pointer -felide-constructors -fno-exceptions
-fno-rtti" ./configure --prefix=/usr/local/mysql --with-low-memory
```

SunOS 5.6 i86pc with `gcc` 2.8.1

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

Linux 2.0.33 i386 with `pgcc` 2.90.29 (`egcs` 1.0.3a)

```
CFLAGS="-O6 -mpentium -mstack-align-double -fomit-frame-pointer"
CXX=gcc CXXFLAGS="-O6 -mpentium -mstack-align-double -fomit-
frame-pointer -felide-constructors -fno-exceptions -fno-rtti"
./configure --prefix=/usr/local/mysql --enable-assembler
--with-mysqld-ldflags=-all-static
```

SCO 3.2v5.0.4 i386 with `gcc` 2.7-95q4

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

AIX 2 4 with `gcc` 2.7.2.2

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

OSF1 V4.0 564 alpha with `gcc` 2.8.1

```
CC=gcc CFLAGS=-O CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
--with-low-memory
```

Irix 6.3 IP32 with `gcc` 2.8.0

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

BSDI BSD/OS 3.1 i386 with `gcc` 2.7.2.1

```
CC=gcc CXX=gcc CXXFLAGS=-O ./configure --prefix=/usr/local/mysql
```

BSDI BSD/OS 2.1 i386 with `gcc` 2.7.2

```
CC=gcc CXX=gcc CXXFLAGS=-O3 ./configure --prefix=/usr/local/mysql
```

Anyone who has more optimal options for any of the configurations listed above can always mail them to the developer's mailing list at `developer@lists.mysql.com`.

RPM distributions prior to **MySQL** 3.22 are user-contributed. Beginning with 3.22, some RPMs are TcX-generated.

4.15 Post-installation setup and testing

Once you've installed **MySQL** (from either a binary or source distribution), you need to initialize the grant tables, start the server and make sure that the server works okay. You may also wish to arrange for the server to be started and stopped automatically when your system starts up and shuts down.

Normally you install the grant tables and start the server like this:

```
shell> cd mysql_installation_directory
shell> ./bin/mysql_install_db
shell> ./bin/safe_mysqld &
```

Testing is most easily done from the top-level directory of the **MySQL** distribution. For a binary distribution, this is your installation directory (typically something like '`/usr/local/mysql`'). For a source distribution, this is the main directory of your **MySQL** source tree.

In the commands shown below in this section and in the following subsections, `BINDIR` is the path to the location in which programs like `mysqladmin` and `safe_mysqld` are installed. For a binary distribution, this is the '`bin`' directory within the distribution. For a source distribution, `BINDIR` is probably '`/usr/local/bin`', unless you specified an installation directory other than '`/usr/local`' when you ran `configure`. `EXECDIR` is the location in which the `mysqld` server is installed. For a binary distribution, this is the same as `BINDIR`. For a source distribution, `EXECDIR` is probably '`/usr/local/libexec`'.

Testing is described in detail below:

1. If necessary, start the `mysqld` server and set up the initial **MySQL** grant tables containing the privileges that determine how users are allowed to connect to the server. This is normally done with the `mysql_install_db` script:

   ```
   shell> scripts/mysql_install_db
   ```

   Typically, `mysql_install_db` needs to be run only the first time you install **MySQL**. Therefore, if you are upgrading an existing installation, you can skip this step. (However, `mysql_install_db` is quite safe to use and will not update any tables that already exist, so if you are unsure what to do, you can always run `mysql_install_db`.)

   `mysql_install_db` creates six tables (`user`, `db`, `host`, `tables_priv`, `columns_priv` and `func`) in the `mysql` database. A description of the initial privileges is given in Section 6.10 [Default privileges], page 106. Briefly, these privileges allow the **MySQL**

`root` user to do anything, and allow anybody to create or use databases with a name of `'test'` or starting with `'test_'`.

If you don't set up the grant tables, the following error will appear in the log file when you start the server:

```
mysqld: Can't find file: 'host.frm'
```

The above may also happens with a binary **MySQL** distribution if you don't start **MySQL** by executing exactly `./bin/safe_mysqld`!

You might need to run `mysql_install_db` as `root`. However, if you prefer, you can run the **MySQL** server as an unprivileged (non-`root`) user, provided that user can read and write files in the database directory. Instructions for running **MySQL** as an unprivileged user are given in Section 18.8 [Changing **MySQL** user], page 314.

If you have problems with `mysql_install_db`, see Section 4.15.1 [`mysql_install_db`], page 75.

There are some alternatives to running the `mysql_install_db` script as it is provided in the **MySQL** distribution:

- You may want to edit `mysql_install_db` before running it, to change the initial privileges that are installed into the grant tables. This is useful if you want to install **MySQL** on a lot of machines with the same privileges. In this case you probably should need only to add a few extra `INSERT` statements to the `mysql.user` and `mysql.db` tables!
- If you want to change things in the grant tables after installing them, you can run `mysql_install_db`, then use `mysql -u root mysql` to connect to the grant tables as the **MySQL** `root` user and issue SQL statements to modify the grant tables directly.
- It is possible to recreate the grant tables completely after they have already been created. You might want to do this if you've already installed the tables but then want to recreate them after editing `mysql_install_db`.

For more information about these alternatives, see Section 6.10 [Default privileges], page 106.

2. Start the **MySQL** server like this:

```
shell> cd mysql_installation_directory
shell> bin/safe_mysqld &
```

If you have problems starting the server, see Section 4.15.2 [Starting server], page 77.

3. Use `mysqladmin` to verify that the server is running. The following commands provide a simple test to check that the server is up and responding to connections:

```
shell> BINDIR/mysqladmin version
shell> BINDIR/mysqladmin variables
```

The output from `mysqladmin version` varies slightly depending on your platform and version of **MySQL**, but should be similar to that shown below:

```
shell> BINDIR/mysqladmin version
mysqladmin  Ver 6.3 Distrib 3.22.9-beta, for pc-linux-gnu on i686
TCX Datakonsult AB, by Monty
```

```
Server version          3.22.9-beta
Protocol version        10
Connection              Localhost via UNIX socket
TCP port                3306
UNIX socket             /tmp/mysql.sock
Uptime:                 16 sec

Running threads: 1  Questions: 20  Reloads: 2  Open tables: 3
```

To get a feeling for what else you can do with `BINDIR/mysqladmin`, invoke it with the `--help` option.

4. Verify that you can shut down the server:

```
shell> BINDIR/mysqladmin -u root shutdown
```

5. Verify that you can restart the server. Do this using `safe_mysqld` or by invoking `mysqld` directly. For example:

```
shell> BINDIR/safe_mysqld --log &
```

If `safe_mysqld` fails, try running it from the **MySQL** installation directory (if you are not already there). If that doesn't work, see Section 4.15.2 [Starting server], page 77.

6. Run some simple tests to verify that the server is working. The output should be similar to what is shown below:

```
shell> BINDIR/mysqlshow
+-----------+
| Databases |
+-----------+
| mysql     |
+-----------+

shell> BINDIR/mysqlshow mysql
Database: mysql
+--------------+
|    Tables    |
+--------------+
| columns_priv |
| db           |
| func         |
| host         |
| tables_priv  |
| user         |
+--------------+

shell> BINDIR/mysql -e "select host,db,user from db" mysql
+------+--------+------+
| host | db     | user |
+------+--------+------+
| %    | test   |      |
| %    | test_% |      |
```

```
+------+--------+------+
```

There is also a benchmark suite in the '`sql-bench`' directory (under the **MySQL** installation directory) that you can use to compare how **MySQL** performs on different platforms. The '`sql-bench/Results`' directory contains the results from many runs against different databases and platforms. To run all tests, execute these commands:

```
shell> cd sql-bench
shell> run-all-tests
```

If you don't have the '`sql-bench`' directory, you are probably using an RPM for a binary distribution. (Source distribution RPMs include the benchmark directory.) In this case, you must first install the benchmark suite before you can use it. Beginning with **MySQL** 3.22, there are benchmark RPM files named '`mysql-bench-VERSION-i386.rpm`' that contain benchmark code and data.

If you have a source distribution, you can also run the tests in the '`tests`' subdirectory. For example, to run '`auto_increment.tst`', do this:

```
shell> BINDIR/mysql -vvf test < ./tests/auto_increment.tst
```

The expected results are shown in the '`./tests/auto_increment.res`' file.

4.15.1 Problems running `mysql_install_db`

This section lists problems you might encounter when you run `mysql_install_db`:

`mysql_install_db` doesn't install the grant tables

You may find that `mysql_install_db` fails to install the grant tables and terminates after displaying the following messages:

```
starting mysqld daemon with databases from XXXXXX
mysql daemon ended
```

In this case, you should examine the log file very carefully! The log should be located in the directory '`XXXXXX`' named by the error message, and should indicate why `mysqld` didn't start. If you don't understand what happened, include the log when you post a bug report using `mysqlbug`! See Section 2.3 [Bug reports], page 15.

There is already a `mysqld` daemon running

In this case, you have probably don't have to run `mysql_install_db` at all. You have to run `mysql_install_db` only once, when you install **MySQL** the first time.

Installing a second `mysqld` daemon doesn't work when one daemon is running

This can happen when you already have an existing **MySQL** installation, but want to put a new installation in a different place (e.g., for testing, or perhaps you simply want to run two installations at the same time). Generally the problem that occurs when you try to run the second server is that it tries to use the same socket and port as the old one. In this case you will get the error message: `Can't start server: Bind on TCP/IP port: Address already in use` or `Can't start server : Bind on unix socket...` You can start the new server with a different socket and port as follows:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> scripts/mysql_install_db
shell> bin/safe_mysqld &
```

After this, you should edit your server boot script to start both daemons with different sockets and ports. For example, it could invoke `safe_mysqld` twice, but with different `--socket`, `--port` and `--basedir` options for each invocation.

You don't have write access to '/tmp'

If you don't have write access to create a socket file at the default place (in '`/tmp`') or permission to create temporary files in '`/tmp`,' you will get an error when running `mysql_install_db` or when starting or using `mysqld`.

You can specify a different socket and temporary directory as follows:

```
shell> TMPDIR=/some_tmp_dir/
shell> MYSQL_UNIX_PORT=/some_tmp_dir/mysqld.sock
shell> export TMPDIR MYSQL_UNIX_PORT
```

'`some_tmp_dir`' should be the path to some directory for which you have write permission.

After this you should be able to run `mysql_install_db` and start the server with these commands:

```
shell> scripts/mysql_install_db
shell> BINDIR/safe_mysqld &
```

`mysqld` crashes immediately

If you are running RedHat 5.0 with a version of `glibc` older than 2.0.7-5, you should make sure you have installed all `glibc` patches! There is a lot of information about this in the **MySQL** mail archives. Links to the mail archives are available at the online **MySQL** documentation page. Also, see Section 4.11.5 [Linux], page 54.

You can also start `mysqld` manually using the `--skip-grant` option and add the privilege information yourself using `mysql`:

```
shell> BINDIR/safe_mysqld --skip-grant &
shell> BINDIR/mysql -u root mysql
```

From `mysql`, manually execute the SQL commands in `mysql_install_db`. Make sure you run `mysqladmin reload` afterward to tell the server to reload the grant tables.

4.15.2 Problems starting the MySQL server

Generally, you start the `mysqld` server in one of three ways:

- By invoking `mysql.server`. This script is used primarily at system startup and shutdown, and is described more fully in Section 4.15.3 [Automatic start], page 78.
- By invoking `safe_mysqld`, which tries to determine the proper options for `mysqld` and then runs it with those options.

- By invoking `mysqld` directly.

Whichever method you use to start the server, if it fails to start up correctly, check the log file to see if you can find out why. Log files are located in the data directory (typically '`/usr/local/mysql/data`' for a binary distribution, '`/usr/local/var`' for a source distribution). Look in the data directory for files with names of the form '`host_name.err`' and '`host_name.log`' where `host_name` is the name of your server host. Then check the last few lines of these files:

```
shell> tail host_name.err
shell> tail host_name.log
```

When the `mysqld` daemon starts up, it changes directory to the data directory. This is where it expects to write log files and the pid (process ID) file, and where it expects to find databases.

The data directory location is hardwired in when the distribution is compiled. However, if `mysqld` expects to find the data directory somewhere other than where it really is on your system, it will not work properly. If you have problems with incorrect paths, you can find out what options `mysqld` allows and what the default path settings are by invoking `mysqld` with the `--help` option. You can override the defaults by specifying the correct pathnames as command-line arguments to `mysqld`. (These options can be used with `safe_mysqld` as well.)

Normally you should need to tell `mysqld` only the base directory under which **MySQL** is installed. You can do this with the `--basedir` option. You can also use `--help` to check the effect of changing path options (note that `--help` *must* be the final option of the `mysqld` command). For example:

```
shell> EXECDIR/mysqld --basedir=/usr/local --help
```

Once you determine the path settings you want, start the server without the `--help` option.

If you get the following error, it means that some other program (or another `mysqld` server) is already using the TCP/IP port or socket `mysqld` is trying to use:

```
Can't start server: Bind on TCP/IP port: Address already in use
  or
Can't start server : Bind on unix socket...
```

Use `ps` to make sure that you don't have another `mysqld` server running. If you can't find another server running, you can try to execute the command `telnet your-host-name tcp-ip-port-number` and press `RETURN` a couple of times. If you don't get a error message like `telnet: Unable to connect to remote host: Connection refused`, something is using the TCP/IP port `mysqld` is trying to use. See Section 4.15.1 [mysql_install_db], page 75, and Section 19.3 [Multiple servers], page 322.

The `safe_mysqld` script is written so that it normally is able to start a server that was installed from either a source or a binary version of **MySQL**, even if these install the server in slightly different locations. `safe_mysqld` expects one of these conditions to be true:

- The server and databases can be found relative to the directory from which `safe_mysqld` is invoked. `safe_mysqld` looks under its working directory for '`bin`' and '`data`' directories (for binary distributions) or for '`libexec`' and '`var`' directories (for source distributions). This condition should be met if you execute `safe_mysqld` from your

MySQL installation directory (for example, '`/usr/local/mysql`' for a binary distribution).

- If the server and databases cannot be found relative to its working directory, `safe_mysqld` attempts to locate them by absolute pathnames. Typical locations are '`/usr/local/libexec`' and '`/usr/local/var`'. The actual locations are determined when the distribution was built from which `safe_mysqld` comes. They should be correct if **MySQL** was installed in a standard location.

Since `safe_mysqld` will try to find the server and databases relative to its own working directory, you can install a binary distribution of **MySQL** anywhere, as long as you start `safe_mysqld` from the **MySQL** installation directory:

```
shell> cd mysql_installation_directory
shell> bin/safe_mysqld &
```

If `safe_mysqld` fails, even when invoked from the **MySQL** installation directory, you can modify it to use the path to `mysqld` and the pathname options that are correct for your system. Note that if you upgrade **MySQL** in the future, your modified version of `safe_mysqld` will be overwritten, so you should make a copy of your edited version that you can reinstall.

If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

```
shell> mysqladmin variables

or

shell> mysqladmin -h 'your-host-name' variables
```

If `safe_mysqld` starts the server but you can't connect to it, you should make sure you have an entry in '`/etc/hosts`' that looks like this:

```
127.0.0.1       localhost
```

This problem occurs only on systems that don't have a working thread library and for which **MySQL** must be configured to use MIT-pthreads.

4.15.3 Starting and stopping MySQL automatically

The `mysql.server` script can be used to start or stop the server, by invoking it with `start` or `stop` arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

`mysql.server` can be found in the '`share/mysql`' directory under the **MySQL** installation directory, or in the '`support-files`' directory of the **MySQL** source tree.

Before `mysql.server` starts the server, it changes directory to the **MySQL** installation directory, then invokes `safe_mysqld`. You might need to edit `mysql.server` if you have a binary distribution that you've installed in a non-standard location. Modify it to `cd` into the proper directory before it runs `safe_mysqld`. If you want the server to run as some specific user, you can change the `mysql_daemon_user=root` line to use another user. You can also modify `mysql.server` to pass other options to `safe_mysqld`.

`mysql.server stop` brings down server by sending a signal to it. You can take down the server manually by executing `mysqladmin shutdown`.

You might want to add these start and stop commands to the appropriate places in your '`/etc/rc*`' files when you start using **MySQL** for production applications. Note that if you modify `mysql.server`, then if you upgrade **MySQL** sometime, your modified version will be overwritten, so you should make a copy of your edited version that you can reinstall.

If your system uses '`/etc/rc.local`' to start external scripts, you should append the following to it:

```
/bin/sh -c 'cd /usr/local/mysql ; ./bin/safe_mysqld &'
```

You can also add options or `mysql.server` in a global '`/etc/my.cnf`' file. A typical '`/etc/my.cnf`' file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/tmp/mysqld.sock
port=3306

[mysql.server]
user=mysql
basedir=/usr/local/mysql
```

The `mysql.server` script uses the following variables: `user`, `datadir`, `basedir`, `bindir` and `pid-file`.

See Section 4.15.4 [Option files], page 79.

4.15.4 Option files

MySQL 3.22 can read default startup options for the server and for clients from option files.

MySQL reads default options from the following files on Unix:

Filename	Purpose
`/etc/my.cnf`	Global options
`DATADIR/my.cnf`	Server-specific options
`~/.my.cnf`	User-specific options

`DATADIR` is the **MySQL** data directory (typically '`/usr/local/mysql/data`' for a binary installation, or '`/usr/local/var`' for a source installation). Note that this is the directory that was specified at configuration time, not the one specified with `--datadir` when `mysqld` starts up! (`--datadir` has no effect on where the server looks for option files, because it looks for them before it processes any command-line arguments.)

MySQL reads default options from the following files on Win32:

Filename	Purpose
`windows-system-directory\my.ini`	
`C:\my.cnf`	Global options
`C:\mysql\data\my.cnf`	Server-specific options

Note that you on Win32 should specify all paths with `/` instead of `\`. If you use `\`, you need to specify this twice, as `\` is the escape character in **MySQL**.

MySQL tries to read option files in the order listed above. If multiple option files exist, an option specified in a file read later takes precedence over the same option specified in a file read earlier. Options specified on the command line take precedence over options specified in any option file. Some options can be specified using environment variables. Options specified on the command line or in option files take precedence over environment variable values.

The following programs support option files: `mysql`, `mysqladmin`, `mysqld`, `mysqldump`, `mysqlimport`, `mysql.server`, `isamchk` and `pack_isam`.

You can use option files to specify any long option that a program supports! Run the program with `--help` to get a list of available options.

An option file can contain lines of the following forms:

`#comment` Comment lines starts with '`#`' or '`;`'. Empty lines are ignored.

`[group]` `group` is the name of the program or group for which you want to set options. After a group line, any `option` or `set-variable` lines apply to the named group until the end of the option file or another group line is given.

`option` This is equivalent to `--option` on the command line.

`option=value`
This is equivalent to `--option=value` on the command line.

`set-variable = variable=value`
This is equivalent to `--set-variable variable=value` on the command line. This syntax must be used to set a `mysqld` variable.

The `client` group allows you to specify options that apply to all **MySQL** clients (not `mysqld`). This is the perfect group to use to specify the password you use to connect to the server. (But make sure the option file is readable and writable only to yourself.)

Note that for options and values, all leading and trailing blanks are automatically deleted. You may use the escape sequences '`\b`', '`\t`', '`\n`', '`\r`', '`\\`' and '`\s`' in your value string ('`\s`' == blank).

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
set-variable = key_buffer=16M
set-variable = max_allowed_packet=1M

[mysqldump]
quick
```

Here is typical user option file:

```
[client]
# The following password will be sent to all standard MySQL clients
password=my_password

[mysql]
no-auto-rehash
```

If you have a source distribution, you will find a sample configuration file named '`my-example.cnf`' in the '`support-files`' directory. If you have a binary distribution, look in the '`DIR/share/mysql`' directory, where `DIR` is the pathname to the **MySQL** installation directory (typically '`/usr/local/mysql`'). You can copy '`my-example.cnf`' to your home directory (rename the copy to '`.my.cnf`') to experiment with.

To tell a **MySQL** program not to read any option files, specify `--no-defaults` as the first option on the command line. This **MUST** be the first option or it will have no effect! If you want to check which options are used, you can give the option `--print-defaults` as the first option.

If you want to force the use of a specific config file, you can use the option `--defaults-file=full-path-to-default-file`. If you do this, only the specified file will be read.

Note for developers: Option file handling is implemented simply by processing all matching options (i.e., options in the appropriate group) before any command line arguments. This works nicely for programs that use the last instance of an option that is specified multiple times. If you have an old program that handles multiply-specified options this way but doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard **MySQL** clients to see how to do this.

4.16 Is there anything special to do when upgrading/downgrading MySQL?

You can always move the **MySQL** form and data files between different versions on the same architecture as long as you have the same base version of **MySQL**. The current base version is 3. If you change the character set by recompiling **MySQL** (which may also change the sort order), you must run `isamchk -r -q` on all tables. Otherwise your indexes may not be ordered correctly.

If you are paranoid and/or afraid of new versions, you can always rename your old `mysqld` to something like `mysqld`-'old-version-number'. If your new `mysqld` then does something unexpected, you can simply shut it down and restart with your old `mysqld`!

When you do an upgrade you should also backup your old databases, of course. Sometimes it's good to be a little paranoid!

After an upgrade, if you experience problems with recompiled client programs, like `Commands out of sync` or unexpected core dumps, you probably have used an old header or library file when compiling your programs. In this case you should check the date for your '`mysql.h`' file and '`libmysqlclient.a`' library to verify that they are from the new **MySQL** distribution. If not, please recompile your programs!

If you get some problems that the new `mysqld` server doesn't want to start or that you can't connect without a password, check that you don't have some old '`my.cnf`' file from

your old installation! You can check this with: `program-name --print-defaults`. If this outputs anything other than the program name, you have a active `my.cnf` file that will may affect things!

It is a good idea to rebuild and reinstall the `Msql-Mysql-modules` distribution whenever you install a new release of **MySQL**, particularly if you notice symptoms such as all your `DBI` scripts dumping core after you upgrade **MySQL**.

4.16.1 Upgrading from a 3.22 version to 3.23

MySQL 3.23 supports tables of the new `MyISAM` type and the old `NISAM` type. You don't have to convert your old tables to use these with 3.23. By default, all new tables will be created with type `MyISAM` (unless you start `mysqld` with the `--default-table-type=isam` option. You can change an `ISAM` table to a `MyISAM` table with `ALTER TABLE` or the Perl script `mysql_convert_table_format`.

3.22 and 3.21 clients will work without any problems with a 3.23 server.

The following lists what you have to watch out for when upgrading to 3.23:

- `INNER` and `DELAYED` are now reserved words.
- `FLOAT(4)` and `FLOAT(8)` are now true floating point types.
- When declaring `DECIMAL(length,dec)` the length argument no longer includes a place for the sign or the decimal point.
- A `TIME` string must now be of one of the following formats: `[[[DAYS] [H]H:]MM:]SS[.fraction]` or `[[[[[H]H]H]H]MM]SS[.fraction]`
- `LIKE` now compares strings using the same character comparison rules as `'='`. If you require the old behavior, you can compile **MySQL** with the `CXXFLAGS=-DLIKE_CMP_TOUPPER` flag.
- When you check/repair tables you should use `myisamchk` for `MyISAM` tables (`.MYI`) and `isamchk` for ISAM (`.ISM`) tables.
- If you want your `mysqldump`s to be compatible between **MySQL** 3.22 and 3.23, you should not use the `--opt` or `--full` option to `mysqldump`.
- Check all your calls to `DATE_FORMAT()` to make sure there is a '`%`' before each format character.
- `mysql_fetch_fields_direct` is now a function (it was a macro) and it returns a pointer to a `MYSQL_FIELD` instead of a `MYSQL_FIELD`.
- `mysql_num_fields()` can no longer be used on a `MYSQL*` object (it's now a function that takes `MYSQL_RES*` as an argument. You should now use `mysql_field_count()` instead.
- In `MySQL` 3.22, the output of `SELECT DISTINCT ...` was almost always sorted. In 3.23, you must use `GROUP BY` or `ORDER BY` to obtain sorted output.
- `SUM()` now returns `NULL`, instead of 0, if there is no matching rows. This is according to ANSI SQL.
- New restricted words: `CASE, THEN, WHEN, ELSE and END`

4.16.2 Upgrading from a 3.21 version to 3.22

Nothing that affects compatibility has changed between 3.21 and 3.22. The only pitfall is that new tables that are created with `DATE` type columns will use the new way to store the date. You can't access these new fields from an old version of `mysqld`.

After installing **MySQL** 3.22, you should start the new server and then run the `mysql_fix_privilege_tables` script. This will add the new privileges that you need to use the `GRANT` command. If you forget this, you will get `Access denied` when you try to use `ALTER TABLE`, `CREATE INDEX` or `DROP INDEX`. If your **MySQL** root user requires a password, you should give this as an argument to `mysql_fix_privilege_tables`.

The C API interface to `mysql_real_connect()` has changed. If you have an old client program that calls this function, you must place a `0` for the new `db` argument (or recode the client to send the `db` element for faster connections). You must also call `mysql_init()` before calling `mysql_real_connect()`! This change was done to allow the new `mysql_options()` function to save options in the `MYSQL` handler structure.

4.16.3 Upgrading from a 3.20 version to 3.21

If you are running a version older than 3.20.28 and want to switch to 3.21.x, you need to do the following:

You can start the `mysqld` 3.21 server with `safe_mysqld --old-protocol` to use it with clients from the 3.20 distribution. In this case, the new client function `mysql_errno()` will not return any server error, only `CR_UNKNOWN_ERROR`, (but it works for client errors) and the server uses the old password() checking rather than the new one.

If you are **NOT** using the `--old-protocol` option to `mysqld`, you will need to make the following changes:

- All client code must be recompiled. If you are using ODBC, you must get the new **MyODBC** 2.x driver.
- The script `scripts/add_long_password` must be run to convert the `Password` field in the `mysql.user` table to `CHAR(16)`.
- All passwords must be reassigned in the `mysql.user` table (to get 62-bit rather than 31-bit passwords).
- The table format hasn't changed, so you don't have to convert any tables.

MySQL 3.20.28 and above can handle the new `user` table format without affecting clients. If you have a **MySQL** version earlier than 3.20.28, passwords will no longer work with it if you convert the `user` table. So to be safe, you should first upgrade to at least 3.20.28 and then upgrade to 3.21.x.

The new client code works with a 3.20.x `mysqld` server, so if you experience problems with 3.21.x, you can use the old 3.20.x server without having to recompile the clients again.

If you are not using the `--old-protocol` option to `mysqld`, old clients will issue the error message:

```
ERROR: Protocol mismatch. Server Version = 10 Client Version = 9
```

The new Perl DBI/DBD interface also supports the old mysqlperl interface. The only change you have to make if you use mysqlperl is to change the arguments to the connect() function. The new arguments are: host, database, user, password (the user and password arguments have changed places). See Section 20.5.2 [Perl DBI Class], page 368.

The following changes may affect queries in old applications:

- HAVING must now be specified before any ORDER BY clause.
- The parameters to LOCATE() have been swapped.
- There are some new reserved words. The most notable are DATE, TIME and TIMESTAMP.

4.16.4 Upgrading to another architecture

If you are using **MySQL** 3.23, you can copy the .frm, the .MYI and the .MYD files between different architectures that support the same floating point format. (**MySQL** takes care of any byte swapping issues).

The **MySQL** data '*.ISD' and the index files '*.ISM' files) are architecture-dependent and in some case OS-dependent. If you want to move your applications to another machine that has a different architecture or OS than your current machine, you should not try to move a database by simply copying the files to the other machine. Use mysqldump instead.

By default, mysqldump will create a file full of SQL statements. You can then transfer the file to the other machine and feed it as input to the mysql client.

Try mysqldump --help to see what options are available. If you are moving the data to a newer version of **MySQL**, you should use mysqldump --opt with the newer version to get a fast, compact dump.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
shell> mysqladmin -h 'other hostname' create db_name
shell> mysqldump --opt db_name \
        | mysql -h 'other hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use:

```
shell> mysqladmin create db_name
shell> mysqldump -h 'other hostname' --opt --compress db_name \
        | mysql db_name
```

You can also store the result in a file, then transfer the file to the target machine and load the file into the database there. For example, you can dump a database to a file on the source machine like this:

```
shell> mysqldump --quick db_name | gzip > db_name.contents.gz
```

(The file created in this example is compressed.) Transfer the file containing the database contents to the target machine and run these commands there:

```
shell> mysqladmin create db_name
shell> gunzip < db_name.contents.gz | mysql db_name
```

You can also use mysqldump and mysqlimport to accomplish the database transfer. For big tables, this is much faster than simply using mysqldump. In the commands shown below,

`DUMPDIR` represents the full pathname of the directory you use to store the output from `mysqldump`.

First, create the directory for the output files and dump the database:

```
shell> mkdir DUMPDIR
shell> mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the `DUMPDIR` directory to some corresponding directory on the target machine and load the files into **MySQL** there:

```
shell> mysqladmin create db_name           # create database
shell> cat DUMPDIR/*.sql | mysql db_name   # create tables in database
shell> mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

Also, don't forget to copy the `mysql` database, since that's where the grant tables (`user`, `db`, `host`) are stored. You may have to run commands as the **MySQL** `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush-privileges` so that the server reloads the grant table information.

5 How standards-compatible is MySQL?

5.1 MySQL extensions to ANSI SQL92

MySQL includes some extensions that you probably will not find in other SQL databases. Be warned that if you use them, your code will not be portable to other SQL servers. In some cases, you can write code that includes **MySQL** extensions, but is still portable, by using comments of the form `/*! ... */`. In this case, **MySQL** will parse and execute the code within the comment as it would any other **MySQL** statement, but other SQL servers will ignore the extensions. For example:

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

If you add a version number after the `'!'`, the syntax will only be executed if the **MySQL** version is equal or newer than the used version number:

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

The above means that if you have 3.23.02 or newer, then **MySQL** will use the `TEMPORARY` keyword.

MySQL extensions are listed below:

- The field types `MEDIUMINT`, `SET`, `ENUM` and the different `BLOB` and `TEXT` types.
- The field attributes `AUTO_INCREMENT`, `BINARY`, `UNSIGNED` and `ZEROFILL`.
- All string comparisons are case insensitive by default, with sort ordering determined by the current character set (ISO-8859-1 Latin1 by default). If you don't like this, you should declare your columns with the `BINARY` attribute or use the `BINARY` cast, which causes comparisons to be done according to the ASCII order used on the **MySQL** server host.
- **MySQL** maps each database to a directory under the **MySQL** data directory, and tables within a database to filenames in the database directory. This has two implications:
  - Database names and table names are case sensitive in **MySQL** on operating systems that have case sensitive filenames (like most Unix systems). If you have a problem remembering table names, adopt a consistent convention, such as always creating databases and tables using lowercase names.
  - Database, table, index, column or alias names may begin with a digit (but may not consist solely of digits).
  - You can use standard system commands to backup, rename, move, delete and copy tables. For example, to rename a table, rename the '`.ISD`', '`.ISM`' and '`.frm`' files to which the table corresponds.
- In SQL statements, you can access tables from different databases with the `db_name.tbl_name` syntax. Some SQL servers provide the same functionality but call this `User space`. **MySQL** dosen't support tablespaces like in: `create table ralph.my_table...IN my_tablespace`.
- `LIKE` is allowed on numeric columns.
- Use of `INTO OUTFILE` and `STRAIGHT_JOIN` in a `SELECT` statement. See Section 7.11 [SELECT], page 176.

- The `SQL_SMALL_RESULT` option in a `SELECT` statement.
- `EXPLAIN SELECT` to get a description on how tables are joined.
- Use of index names, indexes on a prefix of a field, and use of `INDEX` or `KEY` in a `CREATE TABLE` statement. See Section 7.6 [CREATE TABLE], page 167.
- Use of `TEMPORARY` or `IF NOT EXISTS` with `CREATE TABLE`.
- Use of `COUNT(DISTINCT list)` where 'list' is more than one element.
- Use of `CHANGE col_name`, `DROP col_name` or `DROP INDEX` in an `ALTER TABLE` statement. See Section 7.7 [ALTER TABLE], page 172.
- Use of `IGNORE` in an `ALTER TABLE` statement.
- Use of multiple `ADD`, `ALTER`, `DROP` or `CHANGE` clauses in an `ALTER TABLE` statement.
- Use of `DROP TABLE` with the keywords `IF EXISTS`.
- You can drop multiple tables with a single `DROP TABLE` statement.
- The `LIMIT` clause of the `DELETE` statement.
- The `DELAYED` clause of the `INSERT` and `REPLACE` statements.
- The `LOW_PRIORITY` clause of the `INSERT`, `REPLACE`, `DELETE` and `UPDATE` statements.
- Use of `LOAD DATA INFILE`. In many cases, this syntax is compatible with Oracle's `LOAD DATA INFILE`. See Section 7.15 [LOAD DATA], page 183.
- The `OPTIMIZE TABLE` statement. See Section 7.8 [OPTIMIZE TABLE], page 175.
- The `SHOW` statement. See Section 7.20 [SHOW], page 190.
- Strings may be enclosed by either '`"`' or '`'`', not just by '`'`'.
- Use of the escape '\' character.
- The `SET OPTION` statement. See Section 7.24 [SET OPTION], page 199.
- You don't need to name all selected columns in the `GROUP BY` part. This gives better performance for some very specific, but quite normal queries. See Section 7.3.13 [Group by functions], page 165.
- To make it easier for users that come from other SQL environments, **MySQL** supports aliases for many functions. For example, all string functions support both ANSI SQL syntax and ODBC syntax.
- **MySQL** understands the `||` and `&&` operators to mean logical OR and AND, as in the C programming language. In **MySQL**, `||` and `OR` are synonyms, as are `&&` and `AND`. Because of this nice syntax, **MySQL** doesn't support the ANSI SQL `||` operator for string concatenation; use `CONCAT()` instead. Since `CONCAT()` takes any number of arguments, it's easy to convert use of the `||` operator to **MySQL**.
- `CREATE DATABASE` or `DROP DATABASE`. See Section 7.4 [CREATE DATABASE], page 167.
- The `%` operator is a synonym for `MOD()`. That is, `N % M` is equivalent to `MOD(N,M)`. `%` is supported for C programmers and for compatibility with PostgreSQL.
- The `=`, `<>`, `<=` ,`<`, `>=`,`>`, `<<`, `>>`, `<=>`, `AND`, `OR` or `LIKE` operators may be used in column comparisons to the left of the `FROM` in `SELECT` statements. For example:

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

- The `LAST_INSERT_ID()` function. See Section 20.4.29 [mysql_insert_id()], page 348.

- The `REGEXP` and `NOT REGEXP` extended regular expression operators.
- `CONCAT()` or `CHAR()` with one argument or more than two arguments. (In **MySQL**, these functions can take any number of arguments.)
- The `BIT_COUNT()`, `CASE`, `ELT()`, `FROM_DAYS()`, `FORMAT()`, `IF()`, `PASSWORD()`, `ENCRYPT()`, `md5()`, `ENCODE()`, `DECODE()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `TO_DAYS()`, or `WEEKDAY()` functions.
- Use of `TRIM()` to trim substrings. ANSI SQL only supports removal of single characters.
- The `GROUP BY` functions `STD()`, `BIT_OR()` and `BIT_AND()`.
- Use of `REPLACE` instead of `DELETE + INSERT`. See Section 7.14 [REPLACE], page 182.
- The `FLUSH flush_option` statement.

5.2 MySQL differences compared to ANSI SQL92

We try to make **MySQL** follow the ANSI SQL standard and the ODBC SQL standard, but in some cases **MySQL** does some things differently:

- `--` is only a comment if followed by a white space. See Section 5.3.7 [Missing comments], page 90.
- For `VARCHAR` columns, trailing spaces are removed when the value is stored. See Appendix E [Bugs], page 438.
- In some cases, `CHAR` columns are silently changed to `VARCHAR` columns. See Section 7.6.1 [Silent column changes], page 172.
- Privileges for a table is not automatically revoked when you delete a table. You must explicitly issue a `REVOKE` to revoke privileges for a table. See Section 7.25 [GRANT], page 201.

5.3 Functionality missing from MySQL

The following functionality is missing in the current version of **MySQL**. For a prioritized list indicating when new extensions may be added to **MySQL**, you should consult the online **MySQL** TODO list. That is the latest version of the TODO list in this manual. See Appendix F [TODO], page 439.

5.3.1 Sub-selects

The following will not yet work in **MySQL**:

```
SELECT * FROM table1 WHERE id IN (SELECT id FROM table2);
SELECT * FROM table1 WHERE id NOT IN (SELECT id FROM table2);
```

However, in many cases you can rewrite the query without a sub select:

```
SELECT table1.* FROM table1,table2 WHERE table1.id=table2.id;
SELECT table1.* FROM table1 LEFT JOIN table2 ON table1.id=table2.id where table2.id
```

For more complicated sub queries you can create temporary tables to hold the sub query.

MySQL only supports `INSERT ... SELECT ...` and `REPLACE ... SELECT ...` Independent sub-selects will be probably be available in 3.24.0. You can now use the function `IN()` in other contexts, however.

5.3.2 SELECT INTO TABLE

MySQL doesn't yet support the Oracle SQL extension: `SELECT ... INTO TABLE ...`. **MySQL** supports instead the ANSI SQL syntax `INSERT INTO ... SELECT ...`, which is basically the same thing.

Alternatively, you can use `SELECT INTO OUTFILE...` or `CREATE TABLE ... SELECT` to solve your problem.

5.3.3 Transactions

Transactions are not supported. **MySQL** shortly will support atomic operations, which are like transactions without rollback. With atomic operations, you can execute a group of `INSERT`/`SELECT`/whatever commands and be guaranteed that no other thread will interfere. In this context, you won't usually need rollback. Currently, you can prevent interference from other threads by using the `LOCK TABLES` and `UNLOCK TABLES` commands. See Section 7.23 [LOCK TABLES], page 198.

5.3.4 Stored procedures and triggers

A stored procedure is a set of SQL commands that can be compiled and stored in the server. Once this has been done, clients don't need to keep reissuing the entire query but can refer to the stored procedure. This provides better performance because the query has to be parsed only once and less information needs to be sent between the server and the client. You can also raise the conceptual level by having libraries of functions in the server.

A trigger is a stored procedure that is invoked when a particular event occurs. For example, you can install a stored procedure that is triggered each time a record is deleted from a transaction table and that automatically deletes the corresponding customer from a customer table when all his transactions are deleted.

The planned update language will be able to handle stored procedures, but without triggers. Triggers usually slow down everything, even queries for which they are not needed.

To see when **MySQL** might get stored procedures, see Appendix F [TODO], page 439.

5.3.5 Foreign Keys

Note that foreign keys in SQL are not used to join tables, but are used mostly for checking referential integrity. If you want to get results from multiple tables from a `SELECT` statement, you do this by joining tables!

```
SELECT * from table1,table2 where table1.id = table2.id;
```

See Section 7.12 [JOIN], page 178. See Section 8.3.5 [example-Foreign keys], page 215

The `FOREIGN KEY` syntax in **MySQL** exists only for compatibility with other SQL vendors' `CREATE TABLE` commands; it doesn't do anything. The `FOREIGN KEY` syntax without `ON DELETE ...` is mostly used for documentation purposes. Some ODBC applications may use this to produce automatic `WHERE` clauses, but this is usually easy to override. `FOREIGN KEY` is sometimes used as a constraint check, but this check is unnecessary in practice if rows are inserted into the tables in the right order. **MySQL** only supports these clauses because some applications require them to exist (regardless of whether or not they work!).

In **MySQL**, you can work around the problem of `ON DELETE ...` not being implemented by adding the appropriate `DELETE` statement to an application when you delete records from a table that has a foreign key. In practice this is as quick (in some cases quicker) and much more portable than using foreign keys.

In the near future we will extend the `FOREIGN KEY` implementation so that at least the information will be saved in the table specification file and may be retrieved by `mysqldump` and ODBC.

5.3.5.1 Reasons NOT to use foreign keys

There are so many problems with `FOREIGN KEY`s that we don't know where to start:

- Foreign keys make life very complicated, because the foreign key definitions must be stored in a database and implementing them would destroy the whole "nice approach" of using files that can be moved, copied and removed.
- The speed impact is terrible for `INSERT` and `UPDATE` statements, and in this case almost all `FOREIGN KEY` checks are useless because you usually insert records in the right tables in the right order, anyway.
- There is also a need to hold locks on many more tables when updating one table, because the side effects can cascade through the entire database. It's MUCH faster to delete records from one table first and subsequently delete them from the other tables.
- You can no longer restore a table by doing a full delete from the table and then restoring all records (from a new source or from a backup).
- If you have foreign keys you can't dump and restore tables unless you do so in a very specific order.
- It's very easy to do "allowed" circular definitions that make the tables impossible to recreate each table with a single create statement, even if the definition works and is usable.

The only nice aspect of `FOREIGN KEY` is that it gives ODBC and some other client programs the ability to see how a table is connected and to use this to show connection diagrams and to help in building applicatons.

MySQL will soon store `FOREIGN KEY` definitions so that a client can ask for and receive an answer how the original connection was made. The current '`.frm`' file format does not have any place for it.

5.3.6 Views

MySQL doesn't support views, but this is on the TODO.

5.3.7 '`--`' as the start of a comment

Some other SQL databases use '`--`' to start comments. **MySQL** has '`#`' as the start comment character, even if the `mysql` command line tool removes all lines that start with '`--`'. You can also use the C comment style `/* this is a comment */` with **MySQL**. See Section 7.28 [Comments], page 204.

MySQL 3.23.3 and above supports the '`--`' comment style only if the comment is followed by a space. This is because this degenerate comment style has caused many problems with automatically generated SQL queries that have used something like the following code, where we automatically insert the value of the payment for `!payment!`:

```
UPDATE tbl_name SET credit=credit-!payment!
```

What do you think will happen when the value of `payment` is negative?

Because `1--1` is legal in SQL, we think it is terrible that '`--`' means start comment.

In MySQL 3.23 you can however use: `1-- This is a comment`

The following discussing only concerns you if you are running an **MySQL** version earlier than 3.23:

If you have a SQL program in a text file that contains '`--`' comments you should use:

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

instead of the usual:

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

You can also edit the command file "in place" to change the '`--`' comments to '`#`' comments:

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

Change them back with this command:

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

5.4 What standards does MySQL follow?

Entry level SQL92. ODBC level 0-2.

5.5 How to cope without `COMMIT/ROLLBACK`

MySQL doesn't support `COMMIT-ROLLBACK`. The problem is that handling `COMMIT-ROLLBACK` efficiently would require a completely different table layout than **MySQL** uses today. **MySQL** would also need extra threads that do automatic cleanups on the tables and the disk usage would be much higher. This would make **MySQL** about 2-4 times slower than it is today. **MySQL** is much faster than almost all other SQL databases (typically at least 2-3 times faster). One of the reasons for this is the lack of `COMMIT-ROLLBACK`.

For the moment, we are much more for implementing the SQL server language (something like stored procedures). With this you would very seldom really need `COMMIT-ROLLBACK`. This would also give much better performance.

Loops that need transactions normally can be coded with the help of `LOCK TABLES`, and you don't need cursors when you can update records on the fly.

We have transactions and cursors on the TODO but not quite prioritized. If we implement these, it will be as an option to `CREATE TABLE`. That means that `COMMIT-ROLLBACK` will work only on those tables, so that a speed penalty will be imposed on those table only.

We at TcX have a greater need for a real fast database than a 100% general database. Whenever we find a way to implement these features without any speed loss, we will probably do it. For the moment, there are many more important things to do. Check the TODO

for how we prioritize things at the moment. (Customers with higher levels of support can alter this, so things may be reprioritized.)

The current problem is actually `ROLLBACK`. Without `ROLLBACK`, you can do any kind of `COMMIT` action with `LOCK TABLES`. To support `ROLLBACK`, **MySQL** would have to be changed to store all old records that were updated and revert everything back to the starting point if `ROLLBACK` was issued. For simple cases, this isn't that hard to do (the current `isamlog` could be used for this purpose), but it would be much more difficult to implement `ROLLBACK` for `ALTER/DROP/CREATE TABLE`.

To avoid using `ROLLBACK`, you can use the following strategy:

1. Use `LOCK TABLES ...` to lock all the tables you want to access.
2. Test conditions.
3. Update if everything is okay.
4. Use `UNLOCK TABLES` to release your locks.

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

You can also use functions to update records in a single operation. You can get a very efficient application by using the following techniques:

- Modify fields relative to their current value
- Update only those fields that actually have changed

For example, when we are doing updates to some customer information, we update only the customer data that have changed and test only that none of the changed data, or data that depend on the changed data, have changed compared to the original row. The test for changed data is done with the `WHERE` clause in the `UPDATE` statement. If the record wasn't updated, we give the client a message: `"Some of the data you have changed have been changed by another user"`. Then we show the old row versus the new row in a window, so the user can decide which version of the customer record he should use.

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

```
UPDATE tablename SET pay_back=pay_back+'relative change';

UPDATE customer
  SET
    customer_date='current_date',
    address='new address',
    phone='new phone',
    money_he_owes_us=money_he_owes_us+'new_money'
  WHERE
    customer_id=id AND address='old address' AND phone='old phone';
```

As you can see, this is very efficient and works even if another client has changed the values in the `pay_back` or `money_he_owes_us` columns.

In many cases, users have wanted `ROLLBACK` and/or `LOCK TABLES` for the purpose of managing unique identifiers for some tables. This can be handled much more efficiently by using an `AUTO_INCREMENT` column and either the SQL function `LAST_INSERT_ID()` or the C API function `mysql_insert_id()`. See Section 20.4.29 [mysql_insert_id()], page 348.

At TcX, we have never had any need for row-level locking because we have always been able to code around it. Some cases really need row locking, but they are very few. If you want row-level locking, you can use a flag column in the table and do something like this:

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

MySQL returns 1 for the number of affected rows if the row was found and `row_flag` wasn't already 1 in the original row.

You can think of it as **MySQL** changed the above query to:

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

6 The MySQL access privilege system

MySQL has an advanced but non-standard security/privilege system. This section describes how it works.

6.1 What the privilege system does

The primary function of the **MySQL** privilege system is to authenticate a user connecting from a given host, and to associate that user with **select**, **insert**, **update** and **delete** privileges on a database.

Additional functionality includes the ability to have an anonymous user and to grant privileges for **MySQL**-specific functions such as `LOAD DATA INFILE` and administrative operations.

6.2 MySQL user names and passwords

There are several distinctions between the way user names and passwords are used by **MySQL**, and the way they are used by Unix or Windows:

- User names, as used by **MySQL** for authentication purposes, have nothing to do with Unix user names (login names) or Windows user names. Most **MySQL** clients by default try to log in using the current Unix user name as the **MySQL** user name, but that is for convenience only. Client programs allow a different name to be specified with the `-u` or `--user` options. This means that you can't make a database secure in any way unless all **MySQL** user names have passwords. Anyone may attempt to connect to the server using any name, and they will succeed if they specify any name that doesn't have a password.
- **MySQL** user names can be up to 16 characters long; Unix user names typically are limited to 8 characters.
- **MySQL** passwords have nothing to do with Unix passwords. There is no necessary connection between the password you use to log in to a Unix machine and the password you use to access a database on that machine.
- **MySQL** encrypts passwords using a different algorithm than the one used during the Unix login process. See the descriptions of the `PASSWORD()` and `ENCRYPT()` functions in Section 7.3.12 [Miscellaneous functions], page 162.

6.3 Connecting to the MySQL server

MySQL client programs generally require that you specify connection parameters when you want to access a **MySQL** server: the host you want to connect to, your user name and your password. For example, the `mysql` client can be started like this (optional arguments are enclosed between '`[`' and '`]`'):

```
shell> mysql [-h host_name] [-u user_name] [-pyour_pass]
```

Alternate forms of the `-h`, `-u` and `-p` options are `--host=host_name`, `--user=user_name` and `--password=your_pass`. Note that there is *no space* between `-p` or `--password=` and the password following it.

Note: Specifing a password on the command line is not secure! Any user on your system may then find out your password by typing a command like: `ps auxww`. See Section 4.15.4 [Option files], page 79.

`mysql` uses default values for connection parameters that are missing from the command line:

- The default hostname is `localhost`.
- The default user name is your Unix login name.
- No password is supplied if `-p` is missing.

Thus, for a Unix user `joe`, the following commands are equivalent:

```
shell> mysql -h localhost -u joe
shell> mysql -h localhost
shell> mysql -u joe
shell> mysql
```

Other **MySQL** clients behave similarly.

On Unix systems, you can specify different default values to be used when you make a connection, so that you need not enter them on the command line each time you invoke a client program. This can be done in a couple of ways:

- You can specify connection parameters in the `[client]` section of the '`.my.cnf`' configuration file in your home directory. The relevant section of the file might look like this:

  ```
  [client]
  host=host_name
  user=user_name
  password=your_pass
  ```

  See Section 4.15.4 [Option files], page 79.
- You can specify connection parameters using environment values. The host can be specified using `MYSQL_HOST`. The **MySQL** user name can be specified using `USER`, `LOGNAME` or `LOGIN` (although these variables might already be set to your Unix login name, and it may be unwise to change them). The password can be specified using `MYSQL_PWD` (but this is insecure; see next section).

If connection parameters are specified in multiple ways, values specified on the command line take precedence over values specified in configuration files and environment variables, and values in configuration files take precedence over values in environment variables.

6.4 Keeping your password secure

It is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed below, along with an assessment of the risks of each method:

- Use a `-pyour_pass` or `--password=your_pass` option on the command line. This is convenient but insecure, since your password becomes visible to system status programs (such as `ps`) that may be invoked by other users to display command lines.

(**MySQL** clients typically overwrite the command line argument with zeroes during their initialization sequence, but there is still a brief interval during which the value is visible.)

- Use a `-p` or `--password` option (with no `your_pass` value specified). In this case, the client program solicits the password from the terminal:

  ```
  shell> mysql -u user_name -p
  Enter password: ********
  ```

  The client echoes '`*`' characters to the terminal as you enter your password so that onlookers cannot see it.

  It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs non-interactively, there is no opportunity to enter the password from the terminal.

- Store your password in a configuration file. For example, you can list your password in the `[client]` section of the '`.my.cnf`' file in your home directory:

  ```
  [client]
  password=your_pass
  ```

  If you store your password in '`.my.cnf`', the file should not be group or world readable or writable. Make sure the file's access mode is `400` or `600`.

  See Section 4.15.4 [Option files], page 79.

- You can store your password in the `MYSQL_PWD` environment variable, but this method must be considered extremely insecure and should not be used. Some versions of `ps` include an option to display the environment of running processes; your password will be in plain sight for all to see if you set `MYSQL_PWD`. Even on systems without such a version of `ps`, it is unwise to assume there is no other method to observe process environments.

All in all, the safest methods are to have the client program prompt for the password or to specify the password in a properly-protected '`.my.cnf`' file.

6.5 Privileges provided by MySQL

Privilege information is stored in the `user`, `db`, `host`, `tables_priv` and `columns_priv` tables in the `mysql` database (that is, in the database named `mysql`). The **MySQL** server reads the contents of these tables when it starts up and under the circumstances indicated in Section 6.9 [Privilege changes], page 106.

The names used in this manual to refer to the privileges provided by **MySQL** are shown below, along with the table column name associated with each privilege in the grant tables and the context in which the privilege applies:

Privilege	Column	Context
select	`Select_priv`	tables
insert	`Insert_priv`	tables
update	`Update_priv`	tables

delete	`Delete_priv`	tables
index	`Index_priv`	tables
alter	`Alter_priv`	tables
create	`Create_priv`	databases, tables or indexes
drop	`Drop_priv`	databases or tables
grant	`Grant_priv`	databases or tables
reload	`Reload_priv`	server administration
shutdown	`Shutdown_priv`	server administration
process	`Process_priv`	server administration
file	`File_priv`	file access on server

The **select**, **insert**, **update** and **delete** privileges allow you to perform operations on rows in existing tables in a database.

`SELECT` statements require the **select** privilege only if they actually retrieve rows from a table. You can execute certain `SELECT` statements even without permission to access any of the databases on the server. For example, you could use the `mysql` client as a simple calculator:

```
mysql> SELECT 1+1;
mysql> SELECT PI()*2;
```

The **index** privilege allows you to create or drop (remove) indexes.

The **alter** privilege allows you to use `ALTER TABLE`.

The **create** and **drop** privileges allow you to create new databases and tables, or to drop (remove) existing databases and tables.

Note that if you grant the **drop** privilege for the `mysql` database to a user, that user can drop the database in which the **MySQL** access privileges are stored!

The **grant** privilege allows you to give to other users those privileges you yourself possess.

The **file** privilege gives you permission to read and write files on the server using the `LOAD DATA INFILE` and `SELECT ... INTO OUTFILE` statements. Any user to whom this privilege is granted can read or write any file that the **MySQL** server can read or write.

The remaining privileges are used for administrative operations, which are performed using the `mysqladmin` program. The table below shows which `mysqladmin` commands each administrative privilege allows you to execute:

Privilege	**Commands permitted to privilege holders**
reload	`reload`, `refresh`, `flush-privileges`, `flush-hosts`, `flush-logs`, `flush-tables`
shutdown	`shutdown`
process	`processlist`, `kill`

The `reload` command tells the server to reread the grant tables. The `refresh` command flushes all tables and opens and closes the log files. `flush-privileges` is a synonym for `reload`. The other `flush-*` commands perform functions similar to `refresh` but are more limited in scope, and may be preferable in some instances. For example, if you want to flush just the log files, `flush-logs` is a better choice than `refresh`.

The `shutdown` command shuts down the server.

The `processlist` command displays information about the threads executing within the server. The `kill` command kills server threads. You can always display or kill your own threads, but you need the **process** privilege to display or kill threads initiated by other users.

It is a good idea in general to grant privileges only to those users who need them, but you should exercise particular caution in granting certain privileges:

- The **grant** privilege allows users to give away their privileges to other users. Two users with different privileges and with the **grant** privilege are able to combine privileges.
- The **alter** privilege may be used to subvert the privilege system by renaming tables.
- The **file** privilege can be abused to read any world-readable file on the server into a database table, the contents of which can then be accessed using `SELECT`.
- The **shutdown** privilege can be abused to deny service to other users entirely, by terminating the server.
- The **process** privilege can be used to view the plain text of currently executing queries, including queries that set or change passwords.
- Privileges on the `mysql` database can be used to change passwords and other access privilege information. (Passwords are stored encrypted, so a malicious user cannot simply read them. However, with sufficient privileges, that same user can replace a password with a different one.)

There are some things that you cannot do with the **MySQL** privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.

6.6 How the privilege system works

The **MySQL** privilege system ensures that all users may do exactly the things that they are supposed to be allowed to do. When you connect to a **MySQL** server, your identity is determined by **the host from which you connect** and **the user name you specify**. The system grants privileges according to your identity and **what you want to do**.

MySQL considers both your hostname and user name in identifying you because there is little reason to assume that a given user name belongs to the same person everywhere on the Internet. For example, the user `bill` who connects from `whitehouse.gov` need not be the same person as the user `bill` who connects from `microsoft.com`. **MySQL** handles this by allowing you to distinguish users on different hosts that happen to have the same name: you can grant `bill` one set of privileges for connections from `whitehouse.gov`, and a different set of privileges for connections from `microsoft.com`.

MySQL access control involves two stages:

- Stage 1: The server checks whether or not you are even allowed to connect.
- Stage 2: Assuming you can connect, the server checks each request you issue to see whether or not you have sufficient privileges to perform it. For example, if you try to

select rows from a table in a database or drop a table from the database, the server makes sure you have the **select** privilege for the table or the **drop** privilege for the database.

The server uses the `user`, `db` and `host` tables in the `mysql` database at both stages of access control. The fields in these grant tables are shown below:

Table name	`user`	`db`	`host`
Scope fields	`Host`	`Host`	`Host`
	`User`	`Db`	`Db`
	`Password`	`User`	
Privilege fields	`Select_priv`	`Select_priv`	`Select_priv`
	`Insert_priv`	`Insert_priv`	`Insert_priv`
	`Update_priv`	`Update_priv`	`Update_priv`
	`Delete_priv`	`Delete_priv`	`Delete_priv`
	`Index_priv`	`Index_priv`	`Index_priv`
	`Alter_priv`	`Alter_priv`	`Alter_priv`
	`Create_priv`	`Create_priv`	`Create_priv`
	`Drop_priv`	`Drop_priv`	`Drop_priv`
	`Grant_priv`	`Grant_priv`	`Grant_priv`
	`Reload_priv`		
	`Shutdown_priv`		
	`Process_priv`		
	`File_priv`		

For the second stage of access control (request verification), the server may, if the request involves tables, additionally consult the `tables_priv` and `columns_priv` tables. The fields in these tables are shown below:

Table name	`tables_priv`	`columns_priv`
Scope fields	`Host`	`Host`
	`Db`	`Db`
	`User`	`User`
	`Table_name`	`Table_name`
		`Column_name`
Privilege fields	`Table_priv`	`Column_priv`
	`Column_priv`	
Other fields	`Timestamp`	`Timestamp`
	`Grantor`	

Each grant table contains scope fields and privilege fields.

Scope fields determine the scope of each entry in the tables, i.e., the context in which the entry applies. For example, a `user` table entry with `Host` and `User` values of `'thomas.loc.gov'` and `'bob'` would be used for authenticating connections made to the server by `bob` from the host `thomas.loc.gov`. Similarly, a `db` table entry with `Host`, `User` and `Db` fields of `'thomas.loc.gov'`, `'bob'` and `'reports'` would be used when `bob` connects from the host `thomas.loc.gov` to access the `reports` database.

The `tables_priv` and `columns_priv` tables contain scope fields indicating tables or table/column combinations to which each entry applies.

For access-checking purposes, comparisons of `Host` values are case insensitive. `User`, `Password`, `Db` and `Table_name` values are case sensitive. `Column_name` values are case insensitive in **MySQL** 3.22.12 or later.

Privilege fields indicate the privileges granted by a table entry, that is, what operations can be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. The rules used to do this are described in Section 6.8 [Request access], page 103.

Scope fields are strings, declared as shown below; the default value for each is the empty string:

Field name	Type	
`Host`	`CHAR(60)`	
`User`	`CHAR(16)`	
`Password`	`CHAR(16)`	
`Db`	`CHAR(64)`	(`CHAR(60)` for the `tables_priv` and `columns_priv` tables)

In the `user`, `db` and `host` tables, all privilege fields are declared as `ENUM('N','Y')` — each can have a value of `'N'` or `'Y'`, and the default value is `'N'`.

In the `tables_priv` and `columns_priv` tables, the privilege fields are declared as `SET` fields:

Table name	Field name	Possible set elements
`tables_priv`	`Table_priv`	`'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter'`
`tables_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`columns_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`

Briefly, the server uses the grant tables like this:

- The `user` table scope fields determine whether to allow or reject incoming connections. For allowed connections, the privilege fields indicate the user's global (superuser) privileges.
- The `db` and `host` tables are used together:
  - The `db` table scope fields determine which users can access which databases from which hosts. The privilege fields determine which operations are allowed.
  - The `host` table is used as an extension of the `db` table when you want a given `db` table entry to apply to several hosts. For example, if you want a user to be able to use a database from several hosts in your network, leave the `Host` value empty in the user's `db` table entry, then populate the `host` table with an entry for each of those hosts. This mechanism is described more detail in Section 6.8 [Request access], page 103.
- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: they apply at the table and column level rather than at the database level.

Note that administrative privileges (**reload**, **shutdown**, etc.) are specified only in the `user` table. This is because administrative operations are operations on the server itself and are

not database-specific, so there is no reason to list such privileges in the other grant tables. In fact, only the `user` table need be consulted to determine whether or not you can perform an administrative operation.

The **file** privilege is specified only in the `user` table, too. It is not an administrative privilege as such, but your ability to read or write files on the server host is independent of the database you are accessing.

The `mysqld` server reads the contents of the grant tables once, when it starts up. Changes to the grant tables take effect as indicated in Section 6.9 [Privilege changes], page 106.

When you modify the contents of the grant tables, it is a good idea to make sure that your changes set up privileges the way you want. For help in diagnosing problems, see Section 6.13 [Access denied], page 111. For advice on security issues, Section 6.14 [Security], page 114.

A useful diagnostic tool is the `mysqlaccess` script, which Yves Carlier has provided for the **MySQL** distribution. Invoke `mysqlaccess` with the `--help` option to find out how it works. Note that `mysqlaccess` checks access using only the `user`, `db` and `host` tables. It does not check table- or column-level privileges.

6.7 Access control, stage 1: Connection verification

When you attempt to connect to a **MySQL** server, the server accepts or rejects the connection based on your identity and whether or not you can verify your identity by supplying the correct password. If not, the server denies access to you completely. Otherwise, the server accepts the connection, then enters stage 2 and waits for requests.

Your identity is based on two pieces of information:

- The host from which you connect
- Your **MySQL** user name

Identity checking is performed using the three `user` table scope fields (`Host`, `User` and `Password`). The server accepts the connection only if a `user` table entry matches your hostname and user name, and you supply the correct password.

Values in the `user` table scope fields may be specified as follows:

- A `Host` value may be a hostname or an IP number, or `'localhost'` to indicate the local host.
- You can use the wildcard characters '`%`' and '`_`' in the `Host` field.
- A `Host` value of `'%'` matches any hostname. A blank `Host` value is equivalent to `'%'`. Note that these values match *any host that can create a connection to your server!*
- Wildcard characters are not allowed in the `User` field, but you can specify a blank value, which matches any name. If the `user` table entry that matches an incoming connection has a blank user name, the user is considered to be the anonymous user (the user with no name), rather than the name that the client actually specified. This means that a blank user name is used for all further access checking for the duration of the connection (that is, during stage 2).
- The `Password` field can be blank. This does not mean that any password matches, it means the user must connect without specifying a password.

Non-blank `Password` values represent encrypted passwords. **MySQL** does not store passwords in plaintext form for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the `PASSWORD()` function) and compared to the already-encrypted version stored in the `user` table. If they match, the password is correct.

The examples below show how various combinations of `Host` and `User` values in `user` table entries apply to incoming connections:

`Host` value	`User` value	Connections matched by entry
`'thomas.loc.gov'`	`'fred'`	`fred`, connecting from `thomas.loc.gov`
`'thomas.loc.gov'`	`''`	Any user, connecting from `thomas.loc.gov`
`'%'`	`'fred'`	`fred`, connecting from any host
`'%'`	`''`	Any user, connecting from any host
`'%.loc.gov'`	`'fred'`	`fred`, connecting from any host in the `loc.gov` domain
`'x.y.%'`	`'fred'`	`fred`, connecting from `x.y.net`, `x.y.com`,`x.y.edu`, etc. (this is probably not useful)
`'144.155.166.177'`	`'fred'`	`fred`, connecting from the host with IP address `144.155.166.177`
`'144.155.166.%'`	`'fred'`	`fred`, connecting from any host in the `144.155.166` class C subnet

Since you can use IP wildcard values in the `Host` field (e.g., `'144.155.166.%'` to match every host on a subnet), there is the possibility that someone might try to exploit this capability by naming a host `144.155.166.somewhere.com`. To foil such attempts, **MySQL** disallows matching on hostnames that start with digits and a dot. Thus, if you have a host named something like `1.2.foo.com`, its name will never match the `Host` column of the grant tables. Only an IP number can match an IP wildcard value.

An incoming connection may be matched by more than one entry in the `user` table. For example, a connection from `thomas.loc.gov` by `fred` would be matched by several of the entries just shown above. How does the server choose which entry to use if more than one matches? The server resolves this question by sorting the `user` table after reading it at startup time, then looking through the entries in sorted order when a user attempts to connect. The first matching entry is the one that is used.

`user` table sorting works as follows. Suppose the `user` table looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

When the server reads in the table, it orders the entries with the most-specific `Host` values first (`'%'` in the `Host` column means "any host" and is least specific). Entries with the same `Host` value are ordered with the most-specific `User` values first (a blank `User` value means "any user" and is least specific). The resulting sorted `user` table looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

When a connection is attempted, the server looks through the sorted entries and uses the first match found. For a connection from `localhost` by `jeffrey`, the entries with `'localhost'` in the `Host` column match first. Of those, the entry with the blank user name matches both the connecting hostname and user name. (The `'%'/'jeffrey'` entry would have matched, too, but it is not the first match in the table.)

Here is another example. Suppose the `user` table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| thomas.loc.gov |          | ...
+----------------+----------+-
```

The sorted table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| thomas.loc.gov |          | ...
| %              | jeffrey  | ...
+----------------+----------+-
```

A connection from `thomas.loc.gov` by `jeffrey` is matched by the first entry, whereas a connection from `whitehouse.gov` by `jeffrey` is matched by the second.

A common misconception is to think that for a given user name, all entries that explicitly name that user will be used first when the server attempts to find a match for the connection. This is simply not true. The previous example illustrates this, where a connection from `thomas.loc.gov` by `jeffrey` is first matched not by the entry containing `'jeffrey'` as the `User` field value, but by the entry with no user name!

If you have problems connecting to the server, print out the `user` table and sort it by hand to see where the first match is being made.

6.8 Access control, stage 2: Request verification

Once you establish a connection, the server enters stage 2. For each request that comes in on the connection, the server checks whether you have sufficient privileges to perform it, based on the type of operation you wish to perform. This is where the privilege fields in the grant tables come into play. These privileges can come from any of the `user`, `db`, `host`, `tables_priv` or `columns_priv` tables. The grant tables are manipulated with `GRANT` and

`REVOKE` commands. See Section 7.25 [GRANT], page 201. (You may find it helpful to refer to Section 6.6 [Privileges], page 98, which lists the fields present in each of the grant tables.)

The `user` table grants privileges that are assigned to you on a global basis and that apply no matter what the current database is. For example, if the `user` table grants you the **delete** privilege, you can delete rows from any database on the server host! In other words, `user` table privileges are superuser privileges. It is wise to grant privileges in the `user` table only to superusers such as server or database administrators. For other users, you should leave the privileges in the `user` table set to `'N'` and grant privileges on a database-specific basis only, using the `db` and `host` tables.

The `db` and `host` tables grant database-specific privileges. Values in the scope fields may be specified as follows:

- The wildcard characters '`%`' and '`_`' can be used in the `Host` and `Db` fields of either table.
- A `'%'` `Host` value in the `db` table means "any host." A blank `Host` value in the `db` table means "consult the `host` table for further information."
- A `'%'` or blank `Host` value in the `host` table means "any host."
- A `'%'` or blank `Db` value in either table means "any database."
- A blank `User` value in either table matches the anonymous user.

The `db` and `host` tables are read in and sorted when the server starts up (at the same time that it reads the `user` table). The `db` table is sorted on the `Host`, `Db` and `User` scope fields, and the `host` table is sorted on the `Host` and `Db` scope fields. As with the `user` table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching entries, it uses the first match that it finds.

The `tables_priv` and `columns_priv` tables grant table- and column-specific privileges. Values in the scope fields may be specified as follows:

- The wildcard characters '`%`' and '`_`' can be used in the `Host` field of either table.
- A `'%'` or blank `Host` value in either table means "any host."
- The `Db`, `Table_name` and `Column_name` fields cannot contain wildcards or be blank in either table.

The `tables_priv` and `columns_priv` tables are sorted on the `Host`, `Db` and `User` fields. This is similar to `db` table sorting, although since only the `Host` field may contain wildcards, the sorting is simpler.

The request verification process is described below. (If you are familiar with the access-checking source code, you will notice that the description here differs slightly from the algorithm used in the code. The description is equivalent to what the code actually does; it differs only to make the explanation simpler.)

For administrative requests (**shutdown**, **reload**, etc.), the server checks only the `user` table entry, since that is the only table that specifies administrative privileges. Access is granted if the entry allows the requested operation and denied otherwise. For example, if you want to execute `mysqladmin shutdown` but your `user` table entry doesn't grant the **shutdown** privilege to you, access is denied without even checking the `db` or `host` tables. (Since they contain no `Shutdown_priv` column, there is no need to do so.)

For database-related requests (**insert**, **update**, etc.), the server first checks the user's global (superuser) privileges by looking in the `user` table entry. If the entry allows the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user's database-specific privileges by checking the `db` and `host` tables:

1. The server looks in the `db` table for a match on the `Host`, `Db` and `User` fields. `Host` and `User` are matched to the connecting user's hostname and **MySQL** user name. The `Db` field is matched to the database the user wants to access. If there is no entry for the `Host` and `User`, access is denied.
2. If there is a matching `db` table entry and its `Host` field is not blank, that entry defines the user's database-specific privileges.
3. If the matching `db` table entry's `Host` field is blank, it signifies that the `host` table enumerates which hosts should be allowed access to the database. In this case, a further lookup is done in the `host` table to find a match on the `Host` and `Db` fields. If no `host` table entry matches, access is denied. If there is a match, the user's database-specific privileges are computed as the intersection (*not* the union!) of the privileges in the `db` and `host` table entries, i.e., the privileges that are `'Y'` in both entries. (This way you can grant general privileges in the `db` table entry and then selectively restrict them on a host-by-host basis using the `host` table entries.)

After determining the database-specific privileges granted by the `db` and `host` table entries, the server adds them to the global privileges granted by the `user` table. If the result allows the requested operation, access is granted. Otherwise, the server checks the user's table and column privileges in the `tables_priv` and `columns_priv` tables and adds those to the user's privileges. Access is allowed or denied based on the result.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

It may not be apparent why, if the global `user` entry privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database-, table- and column-specific privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT ... SELECT` statement, you need both **insert** and **select** privileges. Your privileges might be such that the `user` table entry grants one privilege and the `db` table entry grants the other. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either table by itself; the privileges granted by both entries must be combined.

The `host` table can be used to maintain a list of "secure" servers. At TcX, the `host` table contains a list of all machines on the local network. These are granted all privileges.

You can also use the `host` table to indicate hosts that are *not* secure. Suppose you have a machine `public.your.domain` that is located in a public area that you do not consider secure. You can allow access to all hosts on your network except that machine by using `host` table entries like this:

```
+--------------------+----+-
| Host               | Db | ...
+--------------------+----+-
| public.your.domain | %  | ... (all privileges set to 'N')
| %.your.domain      | %  | ... (all privileges set to 'Y')
+--------------------+----+-
```

Naturally, you should always test your entries in the grant tables (e.g., using `mysqlaccess`) to make sure your access privileges are actually set up the way you think they are.

6.9 When privilege changes take effect

When `mysqld` starts, all grant table contents are read into memory and become effective at that point.

Modifications to the grant tables that you perform using `GRANT`, `REVOKE`, or `SET PASSWORD` are noticed by the server immediately.

If you modify the grant tables manually (using `INSERT`, `UPDATE`, etc.), you should execute a `FLUSH PRIVILEGES` statement or run `mysqladmin flush-privileges` to tell the server to reload the grant tables. Otherwise your changes will have *no effect* until you restart the server.

When the server notices that the grant tables have been changed, existing client connections are affected as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect at the next `USE db_name` command.

Global privilege changes and password changes take effect the next time the client connects.

6.10 Setting up the initial MySQL privileges

After installing **MySQL**, you set up the initial access privileges by running `scripts/mysql_install_db`. See Section 4.7.1 [Quick install], page 40. The `scripts/mysql_install_db` script starts up the `mysqld` server, then initializes the grant tables to contain the following set of privileges:

- The **MySQL** `root` user is created as a superuser who can do anything. Connections must be made from the local host.

  Note: The initial `root` password is empty, so anyone can connect as `root` *without a password* and be granted all privileges.
- An anonymous user is created that can do anything with databases that have a name of `'test'` or starting with `'test_'`. Connections must be made from the local host. This means any local user can connect and be treated as the anonymous user.
- Other privileges are denied. For example, normal users can't use `mysqladmin shutdown` or `mysqladmin processlist`.

Note: The default privileges are different for Win32. See Section 4.12.4 [Win32 running], page 66.

Since your installation is initially wide open, one of the first things you should do is specify a password for the **MySQL** `root` user. You can do this as follows (note that you specify the password using the `PASSWORD()` function):

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('new_password')
           WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

You can also use the `SET PASSWORD` statement:

```
shell> mysql -u root mysql
mysql> SET PASSWORD FOR root=PASSWORD('new_password');
```

Another way to set the password is by using the `mysqladmin` command:

```
shell> mysqladmin -u root password new_password
```

Note that if you update the password in the `user` table directly using the first method, you must tell the server to reread the grant tables (with `FLUSH PRIVILEGES`), since the change will go unnoticed otherwise.

Once the `root` password has been set, thereafter you must supply that password when you connect to the server as `root`.

You may wish to leave the `root` password blank so that you don't need to specify it while you perform additional setup or testing, but be sure to set it before using your installation for any real production work.

See the `scripts/mysql_install_db` script to see how it sets up the default privileges. You can use this as a basis to see how to add other users.

If you want the initial privileges to be different than those just described above, you can modify `mysql_install_db` before you run it.

To recreate the grant tables completely, remove all the '`*.frm`', '`*.ISM`' and '`*.ISD`' files in the directory containing the `mysql` database. (This is the directory named '`mysql`' under the database directory, which is listed when you run `mysqld --help`.) Then run the `mysql_install_db` script, possibly after editing it first to have the privileges you want.

NOTE: For **MySQL** versions before 3.22.10, you should NOT delete the '`*.frm`' files. If you accidentally do this, you should copy them back from your **MySQL** distribution before running `mysql_install_db`.

6.11 Adding new user privileges to MySQL

You can add users two different ways: by using `GRANT` statements or by manipulating the **MySQL** grant tables directly. The preferred method is to use `GRANT` statements, because they are more concise and less error-prone.

The examples below show how to use the `mysql` client to set up new users. These examples assume that privileges are set up according to the defaults described in the previous section. This means that to make changes, you must be on the same machine where `mysqld` is running, you must connect as the **MySQL** `root` user, and the `root` user must have the **insert** privilege for the `mysql` database and the **reload** administrative privilege. Also, if you have changed the `root` user password, you must specify it for the `mysql` commands below.

You can add new users by issuing `GRANT` statements:

```
shell> mysql --user=root mysql
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@localhost
           IDENTIFIED BY 'something' WITH GRANT OPTION;
mysql> GRANT ALL PRIVILEGES ON *.* TO monty@"%"
           IDENTIFIED BY 'something' WITH GRANT OPTION;
mysql> GRANT RELOAD,PROCESS ON *.* TO admin@localhost;
mysql> GRANT USAGE ON *.* TO dummy@localhost;
```

These `GRANT` statements set up three new users:

`monty` A full superuser who can connect to the server from anywhere, but who must use a password (`'something'` to do so. Note that we must issue `GRANT` statements for both `monty@localhost` and `monty@"%"`. If we don't add the entry with `localhost`, the anonymous user entry for `localhost` that is created by `mysql_install_db` will take precedence when we connect from the local host, because it has a more specific `Host` field value and thuse comes earlier in the `user` table sort order.

`admin` A user who can connect from `localhost` without a password and who is granted the **reload** and **process** administrative privileges. This allows the user to execute the `mysqladmin reload`, `mysqladmin refresh` and `mysqladmin flush-*` commands, as well as `mysqladmin processlist` . No database-related privileges are granted. They can be granted later by issuing additional `GRANT` statements.

`dummy` A user who can connect without a password, but only from the local host. The global privileges are all set to `'N'` — the `USAGE` privilege type allows you to set up a user with no privileges. It is assumed that you will grant database-specific privileges later.

You can also add the same user access information directly by issuing `INSERT` statements and then telling the server to reload the grant tables:

```
shell> mysql --user=root mysql
mysql> INSERT INTO user VALUES('localhost','monty',PASSWORD('something'),
                'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y')
mysql> INSERT INTO user VALUES('%','monty',PASSWORD('something'),
                'Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y','Y')
mysql> INSERT INTO user SET Host='localhost',User='admin',
                 Reload_priv='Y', Process_priv='Y';
mysql> INSERT INTO user (Host,User,Password)
                        VALUES('localhost','dummy','');
mysql> FLUSH PRIVILEGES;
```

Depending on your **MySQL** version, you may have to use a different number of `'Y'` values above (versions prior to 3.22.11 had fewer privilege columns). For the `admin` user, the more readable extended `INSERT` syntax that is available starting with 3.22.11 is used.

Note that to set up a superuser, you need only create a `user` table entry with the privilege fields set to `'Y'`. No `db` or `host` table entries are necessary.

The privilege columns in the `user` table were not set explicitly in the last `INSERT` statement (for the `dummy` user), so those columns are assigned the default value of `'N'`. This is the same thing that `GRANT USAGE` does.

The following example adds a user `custom` who can connect from hosts `localhost`, `server.domain` and `whitehouse.gov`. He wants to access the `bankaccount` database only from `localhost`, the `expenses` database only from `whitehouse.gov` and the `customer` database from all three hosts. He wants to use the password `stupid` from all three hosts.

To set up this user's privileges using `GRANT` statements, run these commands:

```
shell> mysql --user=root mysql
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
           ON bankaccount.*
           TO custom@localhost
           IDENTIFIED BY 'stupid';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
           ON expenses.*
           TO custom@whitehouse.gov
           IDENTIFIED BY 'stupid';
mysql> GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
           ON customer.*
           TO custom@'%'
           IDENTIFIED BY 'stupid';
```

To set up the user's privileges by modifying the grant tables directly, run these commands (note the `FLUSH PRIVILEGES` at the end):

```
shell> mysql --user=root mysql
mysql> INSERT INTO user (Host,User,Password)
       VALUES('localhost','custom',PASSWORD('stupid'));
mysql> INSERT INTO user (Host,User,Password)
       VALUES('server.domain','custom',PASSWORD('stupid'));
mysql> INSERT INTO user (Host,User,Password)
       VALUES('whitehouse.gov','custom',PASSWORD('stupid'));
mysql> INSERT INTO db
       (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
        Create_priv,Drop_priv)
       VALUES
       ('localhost','bankaccount','custom','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
       (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
        Create_priv,Drop_priv)
       VALUES
       ('whitehouse.gov','expenses','custom','Y','Y','Y','Y','Y','Y');
mysql> INSERT INTO db
       (Host,Db,User,Select_priv,Insert_priv,Update_priv,Delete_priv,
        Create_priv,Drop_priv)
       VALUES('%','customer','custom','Y','Y','Y','Y','Y','Y');
mysql> FLUSH PRIVILEGES;
```

The first three `INSERT` statements add `user` table entries that allow user `custom` to connect from the various hosts with the given password, but grant no permissions to him (all privileges are set to the default value of `'N'`). The next three `INSERT` statements add `db` table entries that grant privileges to `custom` for the `bankaccount`, `expenses` and `customer` databases, but only when accessed from the proper hosts. As usual, when the grant tables

are modified directly, the server must be told to reload them (with `FLUSH PRIVILEGES`) so that the privilege changes take effect.

If you want to give a specific user access from any machine in a given domain, you can issue a `GRANT` statement like the following:

```
mysql> GRANT ...
           ON *.*
           TO myusername@"%.mydomainname.com"
           IDENTIFIED BY 'mypassword';
```

To do the same thing by modifying the grant tables directly, do this:

```
mysql> INSERT INTO user VALUES ('%.mydomainname.com', 'myusername',
           PASSWORD('mypassword'),...);
mysql> FLUSH PRIVILEGES;
```

You can also use `xmysqladmin`, `mysql_webadmin` and even `xmysql` to insert, change and update values in the grant tables. You can find these utilities at the **MySQL** Contrib directory.

6.12 How to set up passwords

The examples in the preceding sections illustrate an important principle: when you store a non-empty password using `INSERT` or `UPDATE` statements, you must use the `PASSWORD()` function to encrypt it. This is because the `user` table stores passwords in encrypted form, not as plaintext. If you forget that fact, you are likely to attempt to set passwords like this:

```
shell> mysql -u root mysql
mysql> INSERT INTO user (Host,User,Password)
       VALUES('%','jeffrey','biscuit');
mysql> FLUSH PRIVILEGES;
```

The result is that the plaintext value `'biscuit'` is stored as the password in the `user` table. When the user `jeffrey` attempts to connect to the server using this password, the `mysql` client encrypts it with `PASSWORD()` and sends the result to the server. The server compares the value in the `user` table (which is the plaintext value `'biscuit'`) to the encrypted password (which is *not* `'biscuit'`). The comparison fails and the server rejects the connection:

```
shell> mysql -u jeffrey -pbiscuit test
Access denied
```

Since passwords must be encrypted when they are inserted in the `user` table, the `INSERT` statement should have been specified like this instead:

```
mysql> INSERT INTO user (Host,User,Password)
       VALUES('%','jeffrey',PASSWORD('biscuit'));
```

You must also use the `PASSWORD()` function when you use `SET PASSWORD` statements:

```
mysql> SET PASSWORD FOR jeffrey@"%" = PASSWORD('biscuit');
```

If you set passwords using the `GRANT ... IDENTIFIED BY` statement or the `mysqladmin password` command, the `PASSWORD()` function is unnecessary. They both take care of encrypting the password for you, so you would specify a password of `'biscuit'` like this:

```
mysql> GRANT USAGE ON *.* TO jeffrey@"%" IDENTIFIED BY 'biscuit';
```

or

```
shell> mysqladmin -u jeffrey password biscuit
```

Note: `PASSWORD()` does not perform password encryption in the same way that Unix passwords are encrypted. You should not assume that if your Unix password and your **MySQL** password are the same, `PASSWORD()` will result in the same encrypted value as is stored in the Unix password file. See Section 6.2 [User names], page 94.

6.13 Causes of `Access denied` errors

If you encounter `Access denied` errors when you try to connect to the **MySQL** server, the list below indicates some courses of action you can take to correct the problem:

- Did you run the `mysql_install_db` script after installing **MySQL**, to set up the initial grant table contents? If not, do so. See Section 6.10 [Default privileges], page 106. Test the initial privileges by executing this command:

  ```
  shell> mysql -u root test
  ```

  The server should let you connect without error. You should also make sure you have a file '`user.ISD`' in the **MySQL** database directory. Ordinarily, this is '`PATH/var/mysql/user.ISD`', where `PATH` is the pathname to the **MySQL** installation root.
- After a fresh installation, you should connect to the server and set up your users and their access permissions:

  ```
  shell> mysql -u root mysql
  ```

  The server should let you connect because the **MySQL** `root` user has no password initially. Since that is also a security risk, setting the `root` password is something you should do while you're setting up your other **MySQL** users.

  If you try to connect as `root` and get this error:

  ```
  Access denied for user: '@unknown' to database mysql
  ```

  this means that you don't have an entry in the `user` table with a `User` column value of `'root'` and that `mysqld` cannot resolve the hostname for your client. In this case, you must restart the server with the `--skip-grant-tables` option and edit your '`/etc/hosts`' or '`\windows\hosts`' file to add a entry for your host.
- If you updated an existing **MySQL** installation from a pre-3.22.11 version to 3.22.11 or later, did you run the `mysql_fix_privilege_tables` script? If not, do so. The structure of the grant tables changed with **MySQL** 3.22.11 when the `GRANT` statement became functional.
- If you make changes to the grant tables directly (using `INSERT` or `UPDATE` statement) and your changes seem to be ignored, remember that you must issue a `FLUSH PRIVILEGES` statement or execute a `mysqladmin flush-privileges` command to cause the server to reread the tables. Otherwise your changes have no effect until the next time the server is restarted. Remember that after you set the `root` password, you won't need to specify it until after you flush the privileges, because the server still won't know you've changed the password yet!

- If your privileges seem to have changed in the middle of a session, it may be that a superuser has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in Section 6.9 [Privilege changes], page 106.
- For testing, start the `mysqld` daemon with the `--skip-grant-tables` option. Then you can change the **MySQL** grant tables and use the `mysqlaccess` script to check whether or not your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to start using the new grant tables. **Note:** Reloading the grant tables overrides the `--skip-grant-tables` option. This allows you to tell the server to begin using the grant tables again without bringing it down and restarting it.
- If you have access problems with a Perl, Python or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -pyour_pass db_name`. If you are able to connect using the `mysql` client, there is a problem with your program and not with the access privileges. (Notice that there is no space between `-p` and the password; you can also use the `--password=your_pass` syntax to specify the password.)
- If you can't get your password to work, remember that you must use the `PASSWORD()` function if you set the password with the `INSERT`, `UPDATE` or `SET PASSWORD` statements. The `PASSWORD()` function is unnecessary if you specify the password using the `GRANT ... INDENTIFIED BY` statement or the `mysqladmin password` command. See Section 6.12 [Passwords], page 110.
- `localhost` is a synonym for your local hostname, and is also the default host to which clients try to connect if you specify no host explicitly. However, connections to `localhost` do not work if you are running on a system that uses MIT-pthreads (`localhost` connections are made using Unix sockets, which are not supported by MIT-pthreads). To avoid this problem on such systems, you should use the `--host` option to name the server host explicitly. This will make a TCP/IP connection to the `mysqld` server. In this case, you must have your real hostname in `user` table entries on the server host. (This is true even if you are running a client program on the same host as the server.)
- If you get an `Access denied` error when trying to connect to the database with `mysql -u user_name db_name`, you may have a problem with the `user` table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

  ```
  mysql> SELECT * FROM user;
  ```

  The result should include an entry with the `Host` and `User` columns matching your computer's hostname and your **MySQL** user name.
- The `Access denied` error message will tell you who you are trying to log in as, the host from which you are trying to connect, and whether or not you were using a password. Normally, you should have one entry in the `user` table that exactly matches the hostname and user name that were given in the error message.
- If you get the following error when you try to connect from a different host than the one on which the **MySQL** server is running, then there is no row in the `user` table that matches that host:

```
Host ... is not allowed to connect to this MySQL server
```

You can fix this by using the command line tool `mysql` (on the server host!) to add a row to the `user` table for the user/hostname combination from which you are trying to connect. If you are not running **MySQL** 3.22 and you don't know the IP number or hostname of the machine from which you are connecting, you should put an entry with `'%'` as the `Host` column value in the `user` table and restart `mysqld` with the `--log` option on the server machine. After trying to connect from the client machine, the information in the **MySQL** log will indicate how you really did connect. (Then replace the `'%'` in the `user` table entry with the actual hostname that shows up in the log. Otherwise, you'll have a system that is insecure.)

- If `mysql -u root test` works but `mysql -h your_hostname -u root test` results in `Access denied`, then you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the user table entry specifies an unqualified hostname, but your system's name resolution routines return a fully-qualified domain name (or vice-versa). For example, if you have an entry with host `'tcx'` in the `user` table, but your DNS tells **MySQL** that your hostname is `'tcx.subnet.se'`, the entry will not work. Try adding an entry to the `user` table that contains the IP number of your host as the `Host` column value. (Alternatively, you could add an entry to the `user` table with a `Host` value that contains a wildcard—for example, `'tcx.%'`. However, use of hostnames ending with '`%`' is *insecure* and is *not* recommended!)
- If `mysql -u user_name test` works but `mysql -u user_name other_db_name` doesn't work, you don't have an entry for `other_db_name` listed in the `db` table.
- If `mysql -u user_name db_name` works when executed on the server machine, but `mysql -u host_name -u user_name db_name` doesn't work when executed on another client machine, you don't have the client machine listed in the `user` table or the `db` table.
- If you can't figure out why you get `Access denied`, remove from the `user` table all entries that have `Host` values containing wildcards (entries that contain '`%`' or '`_`'). A very common error is to insert a new entry with `Host='%'` and `User='some user'`, thinking that this will allow you to specify `localhost` to connect from the same machine. The reason that this doesn't work is that the default privileges include an entry with `Host='localhost'` and `User=''`. Since that entry has a `Host` value `'localhost'` that is more specific than `'%'`, it is used in preference to the new entry when connecting from `localhost`! The correct procedure is to insert a second entry with `Host='localhost'` and `User='some_user'`, or to remove the entry with with `Host='localhost'` and `User=''`.
- If you get the following error, you may have a problem with the `db` or `host` table:

  ```
  Access to database denied
  ```

  If the entry selected from the `db` table has an empty value in the `Host` column, make sure there are one or more corresponding entries in the `host` table specifying which hosts the `db` table entry applies to.

  If you get the error when using the SQL commands `SELECT ... INTO OUTFILE` or `LOAD DATA INFILE`, your entry in the `user` table probably doesn't have the **file** privilege enabled.

- Remember that client programs will use connection parameters specified in configuration files or environment variables. If a client seems to be sending the wrong default connection parameters when you don't specify them on the command line, check your environment and the '`.my.cnf`' file in your home directory. You might also check the system-wide **MySQL** configuration files, though it is far less likely that client connection parameters will be specified there. See Section 4.15.4 [Option files], page 79. If you get `Access denied` when you run a client without any options, make sure you haven't specified an old password in any of your option files! See Section 4.15.4 [Option files], page 79.
- If everything else fails, start the `mysqld` daemon with a debugging option (for example, `--debug=d,general,query`). This will print host and user information about attempted connections, as well as information about each command issued. See Section G.1 [Debugging server], page 444.
- If you have any other problems with the **MySQL** grant tables and feel you must post the problem to the mailing list, always provide a dump of the **MySQL** grant tables. You can dump the tables with the `mysqldump mysql` command. As always, post your problem using the `mysqlbug` script. In some cases you may restart `mysqld` with `--skip-grant-tables` to be able to run `mysqldump`.

6.14 How to make MySQL secure against crackers

When you connect to a **MySQL** server, you should normally use a password. The password is not transmitted in clear text over the connection.

All other information is transferred as text that can be read by anyone that is able to watch the connection. If you are concerned about this, you can use the compressed protocol (in **MySQL** 3.22 and above) to make things much harder. To make things even more secure you should install `ssh` (see `http://www.cs.hut.fi/ssh`). With this, you can get an encrypted TCP/IP connection between a **MySQL** server and a **MySQL** client.

To make a **MySQL** system secure, you should strongly consider the following suggestions:

- Use passwords for all **MySQL** users. Remember that anyone can log in as any other person as simply as `mysql -u other_user db_name` if `other_user` has no password. It is common behavior with client/server applications that the client may specify any user name. You can change the password of all users by editing the `mysql_install_db` script before you run it, or only the password for the **MySQL** `root` user like this:

```
shell> mysql -u root mysql
mysql> UPDATE user SET Password=PASSWORD('new_password')
           WHERE user='root';
mysql> FLUSH PRIVILEGES;
```

- Don't run the **MySQL** daemon as the Unix `root` user. `mysqld` can be run as any user. You can also create a new Unix user `mysql` to make everything even more secure. If you run `mysqld` as another Unix user, you don't need to change the `root` user name in the `user` table, because **MySQL** user names have nothing to do with Unix user names. You can edit the `mysql.server` script to start `mysqld` as another Unix user. Normally this is done with the `su` command. For more details, see Section 18.8 [Changing **MySQL** user], page 314.

- If you put a password for the Unix `root` user in the `mysql.server` script, make sure this script is readable only by `root`.
- Check that the Unix user that `mysqld` runs as is the only user with read/write privileges in the database directories.
- Don't give the **process** privilege to all users. The output of `mysqladmin processlist` shows the text of the currently executing queries, so any user who is allowed to execute that command might be able to see if another user issues an `UPDATE user SET password=PASSWORD('not_secure')` query.

  `mysqld` saves an extra connection for users who have the **process** privilege, so that a **MySQL** `root` user can log in and check things even if all normal connections are in use.
- Don't give the **file** privilege to all users. Any user that has this privilege can write a file anywhere in the file system with the privileges of the `mysqld` daemon! To make this a bit safer, all files generated with `SELECT ... INTO OUTFILE` are readable to everyone, and you can't overwrite existing files.

  The **file** privilege may also be used to read any file accessible to the Unix user that the server runs as. This could be abused, for example, by using `LOAD DATA` to load '`/etc/passwd`' into a table, which can then be read with `SELECT`.
- If you don't trust your DNS, you should use IP numbers instead of hostnames in the grant tables. In principle, the `--secure` option to `mysqld` should make hostnames safe. In any case, you should be very careful about using hostname values that contain wildcards!

The following `mysqld` options affect security:

`--secure` IP numbers returned by the `gethostbyname()` system call are checked to make sure they resolve back to the original hostname. This makes it harder for someone on the outside to get access by simulating another host. This option also adds some sanity checks of hostnames. The option is turned off by default in **MySQL** 3.21 since it sometimes takes a long time to perform backward resolutions. **MySQL** 3.22 caches hostnames and has this option enabled by default.

`--skip-grant-tables`
: This option causes the server not to use the privilege system at all. This gives everyone *full access* to all databases! (You can tell a running server to start using the grant tables again by executing `mysqladmin reload`.)

`--skip-name-resolve`
: Hostnames are not resolved. All `Host` column values in the grant tables must be IP numbers or `localhost`.

`--skip-networking`
: Don't allow TCP/IP connections over the network. All connections to `mysqld` must be made via Unix sockets. This option is unsuitable for systems that use MIT-pthreads, because the MIT-pthreads package doesn't support Unix sockets.

7 MySQL language reference

7.1 Literals: how to write strings and numbers

7.1.1 Strings

A string is a sequence of characters, surrounded by either single quote (‘’’) or double quote (‘"’) characters. Examples:

```
'a string'
"another string"
```

Within a string, certain sequences have special meaning. Each of these sequences begins with a backslash (‘\’), known as the *escape character*. **MySQL** recognizes the following escape sequences:

`\0`	An ASCII 0 (`NUL`) character.
`\n`	A newline character.
`\t`	A tab character.
`\r`	A carriage return character.
`\b`	A backspace character.
`\'`	A single quote (‘’’) character.
`\"`	A double quote (‘"’) character.
`\\`	A backslash (‘\’) character.
`\%`	A ‘`%`’ character. This is used to search for literal instances of ‘`%`’ in contexts where ‘`%`’ would otherwise be interpreted as a wildcard character.
`\_`	A ‘`_`’ character. This is used to search for literal instances of ‘`_`’ in contexts where ‘`_`’ would otherwise be interpreted as a wildcard character.

There are several ways to include quotes within a string:

- A ‘’’ inside a string quoted with ‘’’ may be written as ‘’’’.
- A ‘"’ inside a string quoted with ‘"’ may be written as ‘""’.
- You can precede the quote character with an escape character (‘\’).
- A ‘’’ inside a string quoted with ‘"’ needs no special treatment and need not be doubled or escaped. In the same way, ‘"’ inside a string quoted with ‘’’ needs no special treatment.

The `SELECT` statements shown below demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+
```

```
mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT "This\nIs\nFour\nlines";
+--------------------+
| This
Is
Four
lines |
+--------------------+
```

If you want to insert binary data into a `BLOB` column, the following characters must be represented by escape sequences:

`NUL` ASCII 0. You should represent this by '`\0`' (a backslash and an ASCII '`0`' character).

`\` ASCII 92, backslash. Represent this by '`\\`'.

`'` ASCII 39, single quote. Represent this by '`\'`'.

`"` ASCII 34, double quote. Represent this by '`\"`'.

If you write C code, you can use the C API function `mysql_escape_string()` to escape characters for the `INSERT` statement. See Section 20.3 [C API function overview], page 327. In Perl, you can use the `quote` method of the `DBI` package to convert special characters to the proper escape sequences. See Section 20.5.2 [Perl `DBI` Class], page 368.

You should use an escape function on any string that might contain any of the special characters listed above!

7.1.2 Numbers

Integers are represented as a sequence of digits. Floats use '`.`' as a decimal separator. Either type of number may be preceded by '`-`' to indicate a negative value.

Examples of valid integers:

```
1221
0
-32
```

Examples of valid floating-point numbers:

```
294.42
-32032.6809e+10
148.00
```

An integer may be used in a floating-point context; it is interpreted as the equivalent floating-point number.

7.1.3 Hexadecimal values

MySQL supports hexadecimal values. In number context these acts like an integer (64 bit precision). In string context these acts like a binary string where each pair of hex digits is converted to a character.

```
mysql> SELECT 0xa+0
       -> 10
mysql> select 0x5061756c;
       -> Paul
```

Hexadecimal strings is often used by ODBC to give values for BLOB columns.

7.1.4 `NULL` values

The `NULL` value means "no data" and is different from values such as `0` for numeric types or the empty string for string types. See Section 18.15 [Problems with `NULL`], page 317.

`NULL` may be represented by `\N` when using the text file import or export formats (`LOAD DATA INFILE`, `SELECT ... INTO OUTFILE`). See Section 7.15 [`LOAD DATA`], page 183.

7.1.5 Database, table, index, column and alias names

Database, table, index, column and alias names all follow the same rules in **MySQL**:

- A name may consist of alphanumeric characters from the current character set and also '_' and '$'. The default character set is ISO-8859-1 Latin1; this may be changed by recompiling **MySQL**. See Section 9.1.1 [Character sets], page 240.
- A database, table, index or column name can be up to 64 characters long. An alias name can be up to 256 characters long.
- A name may start with any character that is legal in a name. In particular, a name may start with a number (this differs from many other database systems!). However, a name cannot consist *only* of numbers.
- It is recommended that you do not use names like `1e`, because an expression like `1e+1` is ambiguous. It may be interpreted as the expression `1e + 1` or as the number `1e+1`.
- You cannot use the '.' character in names because it is used to extend the format by which you can refer to columns (see immediately below).

In **MySQL** you can refer to a column using any of the following forms:

Column reference	Meaning
`col_name`	Column `col_name` from whichever table used in the query contains a column of that name
`tbl_name.col_name`	Column `col_name` from table `tbl_name` of the current database
`db_name.tbl_name.col_name`	Column `col_name` from table `tbl_name` of the database `db_name`. This form is available in **MySQL** 3.22 or later.

You need not specify a `tbl_name` or `db_name.tbl_name` prefix for a column reference in a statement unless the reference would be ambiguous. For example, suppose tables `t1` and `t2` each contain a column `c`, and you retrieve `c` in a `SELECT` statement that uses both `t1`

and `t2`. In this case, `c` is ambiguous because it is not unique among the tables used in the statement, so you must indicate which table you mean by writing `t1.c` or `t2.c`. Similarly, if you are retrieving from a table `t` in database `db1` and from a table `t` in database `db2`, you must refer to columns in those tables as `db1.t.col_name` and `db2.t.col_name`.

The syntax `.tbl_name` means the table `tbl_name` in the current database. This syntax is accepted for ODBC compatibility, because some ODBC programs prefix table names with a '`.`' character.

7.1.5.1 Case sensitivity in names

In **MySQL**, databases and tables correspond to directories and files within those directories. Consequently, the case sensitivity of the underlying operating system determines the case sensitivity of database and table names. This means database and table names are case sensitive in Unix and case insensitive in Win32.

Note: Although database and table names are case insensitive for Win32, you should not refer to a given database or table using different cases within the same query. The following query would not work because it refers to a table both as `my_table` and as `MY_TABLE`:

```
SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column names are case insensitive in all cases.

Aliases on tables are case sensitive. The following query would not work because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a
           WHERE a.col_name = 1 OR A.col_name = 2;
```

Aliases on columns are case insensitive.

7.2 Column types

MySQL supports a number of column types, which may be grouped into three categories: numeric types, date and time types, and string (character) types. This section first gives an overview of the types available and summarizes the storage requirements for each column type, then provides a more detailed description of the properties of the types in each category. The overview is intentionally brief. The more detailed descriptions should be consulted for additional information about particular column types, such as the allowable formats in which you can specify values.

The column types supported by **MySQL** are listed below. The following code letters are used in the descriptions:

`M` Indicates the maximum display size. The maximum legal display size is 255.

`D` Applies to floating-point types and indicates the number of digits following the decimal point.

Square brackets ('`[`' and '`]`') indicate parts of type specifiers that are optional.

Note that if you specify `ZEROFILL` for a column, **MySQL** will automatically add the `UNSIGNED` attribute to the column.

`TINYINT[(M)] [UNSIGNED] [ZEROFILL]`
: A very small integer. The signed range is `-128` to `127`. The unsigned range is `0` to `255`.

`SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`
: A small integer. The signed range is `-32768` to `32767`. The unsigned range is `0` to `65535`.

`MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`
: A medium-size integer. The signed range is `-8388608` to `8388607`. The unsigned range is `0` to `16777215`.

`INT[(M)] [UNSIGNED] [ZEROFILL]`
: A normal-size integer. The signed range is `-2147483648` to `2147483647`. The unsigned range is `0` to `4294967295`.

`INTEGER[(M)] [UNSIGNED] [ZEROFILL]`
: This is a synonym for `INT`.

`BIGINT[(M)] [UNSIGNED] [ZEROFILL]`
: A large integer. The signed range is `-9223372036854775808` to `9223372036854775807`. The unsigned range is `0` to `18446744073709551615`. Note that all arithmetic is done using signed `BIGINT` or `DOUBLE` values, so you shouldn't use unsigned big integers larger than `9223372036854775807` (63 bits) except with bit functions! Note that `-`, `+` and `*` will use `BIGINT` arithmetic when both arguments are `INTEGER` values! This means that if you multiply two big integers (or results from functions that return integers) you may get unexpected results if the result is larger than `9223372036854775807`.

`FLOAT(precision) [ZEROFILL]`
: A floating-point number. Cannot be unsigned. `precision` can be `4` or `8`. `FLOAT(4)` is a single-precision number and `FLOAT(8)` is a double-precision number. These types are like the `FLOAT` and `DOUBLE` types described immediately below. `FLOAT(4)` and `FLOAT(8)` have the same ranges as the corresponding `FLOAT` and `DOUBLE` types, but their display size and number of decimals is undefined.

  In **MySQL** 3.23, this is a true floating point value. In earlier **MySQL** versions, `FLOAT(precision)` always has 2 decimals.

  This syntax is provided for ODBC compatibility.

`FLOAT[(M,D)] [ZEROFILL]`
: A small (single-precision) floating-point number. Cannot be unsigned. Allowable values are `-3.402823466E+38` to `-1.175494351E-38`, `0` and `1.175494351E-38` to `3.402823466E+38`.

`DOUBLE[(M,D)] [ZEROFILL]`
: A normal-size (double-precision) floating-point number. Cannot be unsigned. Allowable values are `-1.7976931348623157E+308` to `-2.2250738585072014E-308`, `0` and `2.2250738585072014E-308` to `1.7976931348623157E+308`.

`DOUBLE PRECISION[(M,D)] [ZEROFILL]`
`REAL[(M,D)] [ZEROFILL]`
: These are synonyms for `DOUBLE`.

`DECIMAL(M,D) [ZEROFILL]`
: An unpacked floating-point number. Cannot be unsigned. Behaves like a `CHAR` column: "unpacked" means the number is stored as a string, using one character for each digit of the value, the decimal point, and, for negative numbers, the '`-`' sign. If `D` is 0, values will have no decimal point or fractional part. The maximum range of `DECIMAL` values is the same as for `DOUBLE`, but the actual range for a given `DECIMAL` column may be constrained by the choice of `M` and `D`.

  In **MySQL** 3.23 the `M` argument no longer includes the sign or the decimal point. (This is according to ANSI SQL.)

`NUMERIC(M,D) [ZEROFILL]`
: This is a synonym for `DECIMAL`.

`DATE`
: A date. The supported range is `'1000-01-01'` to `'9999-12-31'`. **MySQL** displays `DATE` values in `'YYYY-MM-DD'` format, but allows you to assign values to `DATE` columns using either strings or numbers.

`DATETIME`
: A date and time combination. The supported range is `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`. **MySQL** displays `DATETIME` values in `'YYYY-MM-DD HH:MM:SS'` format, but allows you to assign values to `DATETIME` columns using either strings or numbers.

`TIMESTAMP[(M)]`
: A timestamp. The range is `'1970-01-01 00:00:00'` to sometime in the year `2037`. **MySQL** displays `TIMESTAMP` values in `YYYYMMDDHHMMSS`, `YYMMDDHHMMSS`, `YYYYMMDD` or `YYMMDD` format, depending on whether `M` is `14` (or missing), `12`, `8` or `6`, but allows you to assign values to `TIMESTAMP` columns using either strings or numbers. A `TIMESTAMP` column is useful for recording the date and time of an `INSERT` or `UPDATE` operation because it is automatically set to the date and time of the most recent operation if you don't give it a value yourself. You can also set it to the current date and time by assigning it a `NULL` value. See Section 7.2.3 [Date and time types], page 125.

`TIME`
: A time. The range is `'-838:59:59'` to `'838:59:59'`. **MySQL** displays `TIME` values in `'HH:MM:SS'` format, but allows you to assign values to `TIME` columns using either strings or numbers.

`YEAR`
: A year. The allowable values are `1901` to `2155`, and `0000`. **MySQL** displays `YEAR` values in `YYYY` format, but allows you to assign values to `YEAR` columns using either strings or numbers. (The `YEAR` type is new in **MySQL** 3.22.)

`CHAR(M) [BINARY]`
: A fixed-length string that is always right-padded with spaces to the specified length when stored. The range of `M` is 1 to 255 characters. Trailing spaces are removed when the value is retrieved. `CHAR` values are sorted and compared in case-insensitive fashion unless the `BINARY` keyword is given.

`VARCHAR(M) [BINARY]`
: A variable-length string. Note: Trailing spaces are removed when the value is stored (this differs from the ANSI SQL specification). The range of `M` is 1 to 255 characters. `VARCHAR` values are sorted and compared in case-insensitive fashion unless the `BINARY` keyword is given. See Section 7.6.1 [Silent column changes], page 172.

`TINYBLOB`
`TINYTEXT`
: A `BLOB` or `TEXT` column with a maximum length of 255 (2^8 - 1) characters. See Section 7.6.1 [Silent column changes], page 172.

`BLOB`
`TEXT`
: A `BLOB` or `TEXT` column with a maximum length of 65535 (2^16 - 1) characters. See Section 7.6.1 [Silent column changes], page 172.

`MEDIUMBLOB`
`MEDIUMTEXT`
: A `BLOB` or `TEXT` column with a maximum length of 16777215 (2^24 - 1) characters. See Section 7.6.1 [Silent column changes], page 172.

`LONGBLOB`
`LONGTEXT`
: A `BLOB` or `TEXT` column with a maximum length of 4294967295 (2^32 - 1) characters. See Section 7.6.1 [Silent column changes], page 172.

`ENUM('value1','value2',...)`
: An enumeration. A string object that can have only one value, chosen from the list of values `'value1'`, `'value2'`, `...`, or `NULL`. An `ENUM` can have a maximum of 65535 distinct values.

`SET('value1','value2',...)`
: A set. A string object that can have zero or more values, each of which must be chosen from the list of values `'value1'`, `'value2'`, `...` A `SET` can have a maximum of 64 members.

7.2.1 Column type storage requirements

The storage requirements for each of the column types supported by **MySQL** are listed below by category.

Numeric types

Column type	Storage required
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT	4 bytes
INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(4)	4 bytes
FLOAT(8)	8 bytes
FLOAT	4 bytes
DOUBLE	8 bytes
DOUBLE PRECISION	8 bytes
REAL	8 bytes
DECIMAL(M,D)	M bytes (D+2, if M < D)
NUMERIC(M,D)	M bytes (D+2, if M < D)

Date and time types

Column type	Storage required
DATETIME	8 bytes
DATE	3 bytes
TIMESTAMP	4 bytes
TIME	3 bytes
YEAR	1 byte

String types

Column type	Storage required
CHAR(M)	M bytes, 1 <= M <= 255
VARCHAR(M)	L+1 bytes, where L <= M and 1 <= M <= 255
TINYBLOB, TINYTEXT	L+1 bytes, where L < 2^8
BLOB, TEXT	L+2 bytes, where L < 2^16
MEDIUMBLOB, MEDIUMTEXT	L+3 bytes, where L < 2^24
LONGBLOB, LONGTEXT	L+4 bytes, where L < 2^32
ENUM('value1','value2',...)	1 or 2 bytes, depending on the number of enumeration values (65535 values maximum)
SET('value1','value2',...)	1, 2, 3, 4 or 8 bytes, depending on the number of set members (64 members maximum)

VARCHAR and the BLOB and TEXT types are variable-length types, for which the storage requirements depend on the actual length of column values (represented by L in the preceding table), rather than on the type's maximum possible size. For example, a VARCHAR(10) column can hold a string with a maximum length of 10 characters. The actual storage required is the length of the string (L), plus 1 byte to record the length of the string. For the string 'abcd', L is 4 and the storage requirement is 5 bytes.

The BLOB and TEXT types require 1, 2, 3 or 4 bytes to record the length of the column value, depending on the maximum possible length of the type.

If a table includes any variable-length column types, the record format will also be variable-length. Note that when a table is created, **MySQL** may under certain conditions change a column from a variable-length type to a fixed-length type, or vice-versa. See Section 7.6.1 [Silent column changes], page 172.

The size of an ENUM object is determined by the number of different enumeration values. 1 byte is used for enumerations with up to 255 possible values. 2 bytes are used for enumerations with up to 65535 values.

The size of a SET object is determined by the number of different set members. If the set size is N, the object occupies (N+7)/8 bytes, rounded up to 1, 2, 3, 4 or 8 bytes. A SET can have a maximum of 64 members.

7.2.2 Numeric types

All integer types can have an optional attribute UNSIGNED. Unsigned values can be used when you want to allow only positive numbers in a column and you need a little bigger numeric range for the column.

All numeric types can have an optional attribute ZEROFILL. Values for ZEROFILL columns are left-padded with zeroes up to the maximum display length when they are displayed. For example, for a column declared as INT(5) ZEROFILL, a value of 4 is retrieved as 00004.

When asked to store a value in a numeric column that is outside the column type's allowable range, **MySQL** clips the value to the appropriate endpoint of the range and stores the resulting value instead.

For example, the range of an INT column is -2147483648 to 2147483647. If you try to insert -9999999999 into an INT column, the value is clipped to the lower endpoint of the range, and -2147483648 is stored instead. Similarly, if you try to insert 9999999999, 2147483647 is stored instead.

If the INT column is UNSIGNED, the size of the column's range is the same but its endpoints shift up to 0 and 4294967295. If you try to store -9999999999 and 9999999999, the values stored in the column become 0 and 4294967296.

Conversions that occur due to clipping are reported as "warnings" for ALTER TABLE, LOAD DATA INFILE, UPDATE and multi-row INSERT statements.

The maximum display size (M) and number of decimals (D) are used for formatting and calculation of maximum column width.

MySQL will store any value that fits a column's storage type even if the value exceeds the display size. For example, an INT(4) column has a display size of 4. Suppose you insert a value which has more than 4 digits into the column, such as 12345. The display size is exceeded, but the allowable range of the INT type is not, so **MySQL** stores the actual value, 12345. When retrieving the value from the column, **MySQL** returns the actual value stored in the column.

The DECIMAL type is considered a numeric type (as is its synonym, NUMERIC), but such values are stored as strings. One character is used for each digit of the value, the decimal

point (if `D > 0`) and the '`-`' sign (for negative numbers). If `D` is 0, `DECIMAL` and `NUMERIC` values contain no decimal point or fractional part.

The maximum range of `DECIMAL` values is the same as for `DOUBLE`, but the actual range for a given `DECIMAL` column may be constrained by the choice of `M` and `D`. For example, a type specification such as `DECIMAL(4,2)` indicates a maximum length of four characters with two digits after the decimal point. Due to the way the `DECIMAL` type is stored, this specification results in an allowable range of `-.99` to `9.99`, much less than the range of a `DOUBLE`.

To avoid some rounding problems, **MySQL** always rounds everything that it stores in any floating-point column to the number of decimals indicated by the column specification. Suppose you have a column type of `FLOAT(8,2)`. The number of decimals is 2, so a value such as `2.333` is rounded to two decimals and stored as `2.33`.

7.2.3 Date and time types

The date and time types are `DATETIME`, `DATE`, `TIMESTAMP`, `TIME` and `YEAR`. Each of these has a range of legal values, as well as a "zero" value that is used when you specify an illegal value.

Here are some general considerations to keep in mind when working with date and time types:

- **MySQL** retrieves values for a given date or time type in a standard format, but it attempts to interpret a variety of formats for values that you supply (e.g., when you specify a value to be assigned to or compared to a date or time type). Nevertheless, only the formats described in the following sections are supported. It is expected that you will supply legal values, and unpredictable results may occur if you use values in other formats.
- Although **MySQL** tries to interpret values in several formats, it always expects the year part of date values to be leftmost. Dates must be given in year-month-day order (e.g., `'98-09-04'`), rather than in the month-day-year or day-month-year orders commonly used elsewhere (e.g., `'09-04-98'`, `'04-09-98'`).
- **MySQL** automatically converts a date or time type value to a number if the value is used in a numeric context, and vice versa.
- When **MySQL** encounters a value for a date or time type that is out of range or otherwise illegal for the type, it converts the value to the "zero" value for that type. (The exception is that out-of-range `TIME` values are clipped to the appropriate endpoint of the `TIME` range.) The table below shows the format of the "zero" value for each type:

Column type	**"Zero" value**
`DATETIME`	`'0000-00-00 00:00:00'`
`DATE`	`'0000-00-00'`
`TIMESTAMP`	`00000000000000` (length depends on display size)
`TIME`	`'00:00:00'`
`YEAR`	`0000`

- The "zero" values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values `'0'` or `0`, which are easier to write.

- "Zero" date or time values used through **MyODBC** are converted automatically to `NULL` in **MyODBC** 2.50.12 and above, because ODBC can't handle such values.

7.2.3.1 Y2K issues and date types

MySQL itself is Y2K-safe (see Section 1.6 [Year 2000 compliance], page 7), but input values presented to **MySQL** may not be. Any input containing 2-digit year values is ambiguous, since the century is unknown. Such values must be interpreted into 4-digit form since **MySQL** stores years internally using four digits.

For `DATETIME`, `DATE`, `TIMESTAMP` and `YEAR` types, **MySQL** interprets dates with ambiguous year values using the following rules:

- Year values in the range `00-69` are converted to `2000-2069`.
- Year values in the range `70-99` are converted to `1970-1999`.

Remember that these rules provide only reasonable guesses as to what your data mean. If the heuristics used by **MySQL** don't produce the correct values, you should provide unambiguous input containing 4-digit year values.

7.2.3.2 The `DATETIME`, `DATE` and `TIMESTAMP` types

The `DATETIME`, `DATE` and `TIMESTAMP` types are related. This section describes their characteristics, how they are similar and how they differ.

The `DATETIME` type is used when you need values that contain both date and time information. **MySQL** retrieves and displays `DATETIME` values in `'YYYY-MM-DD HH:MM:SS'` format. The supported range is `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`. ("Supported" means that although earlier values might work, there is no guarantee that they will.)

The `DATE` type is used when you need only a date value, without a time part. **MySQL** retrieves and displays `DATE` values in `'YYYY-MM-DD'` format. The supported range is `'1000-01-01'` to `'9999-12-31'`.

The `TIMESTAMP` column type provides a type that you can use to automatically mark `INSERT` or `UPDATE` operations with the current date and time. If you have multiple `TIMESTAMP` columns, only the first one is updated automatically.

Automatic updating of the first `TIMESTAMP` column occurs under any of the following conditions:

- The column is not specified explicitly in an `INSERT` or `LOAD DATA INFILE` statement.
- The column is not specified explicitly in an `UPDATE` statement and some other column changes value. (Note that an `UPDATE` that sets a column to the value it already has will not cause the `TIMESTAMP` column to be updated, because if you set a column to its current value, **MySQL** ignores the update for efficiency.)
- You explicitly set the `TIMESTAMP` column to `NULL`.

`TIMESTAMP` columns other than the first may also be set to the current date and time. Just set the column to `NULL`, or to `NOW()`.

You can set any TIMESTAMP column to a value different than the current date and time by setting it explicitly to the desired value. This is true even for the first TIMESTAMP column. You can use this property if, for example, you want a TIMESTAMP to be set to the current date and time when you create a row, but not to be changed whenever the row is updated later:

- Let **MySQL** set the column when the row is created. This will initialize it to the current date and time.
- When you perform subsequent updates to other columns in the row, set the TIMESTAMP column explicitly to its current value.

On the other hand, you may find it just as easy to use a DATETIME column that you initialize to NOW() when the row is created and leave alone for subsequent updates.

TIMESTAMP values may range from the beginning of 1970 to sometime in the year 2037, with a resolution of one second. Values are displayed as numbers.

The format in which **MySQL** retrieves and displays TIMESTAMP values depends on the display size, as illustrated by the table below. The 'full' TIMESTAMP format is 14 digits, but TIMESTAMP columns may be created with shorter display sizes:

Column type	**Display format**
TIMESTAMP(14)	YYYYMMDDHHMMSS
TIMESTAMP(12)	YYMMDDHHMMSS
TIMESTAMP(10)	YYMMDDHHMM
TIMESTAMP(8)	YYYYMMDD
TIMESTAMP(6)	YYMMDD
TIMESTAMP(4)	YYMM
TIMESTAMP(2)	YY

All TIMESTAMP columns have the same storage size, regardless of display size. The most common display sizes are 6, 8, 12, and 14. You can specify an arbitrary display size at table creation time, but values of 0 or greater than 14 are coerced to 14. Odd-valued sizes in the range from 1 to 13 are coerced to the next higher even number.

You can specify DATETIME, DATE and TIMESTAMP values using any of a common set of formats:

- As a string in either 'YYYY-MM-DD HH:MM:SS' or 'YY-MM-DD HH:MM:SS' format. A "relaxed" syntax is allowed—any non-numeric character may be used as the delimiter between date parts or time parts. For example, '98-12-31 11:30:45', '98.12.31 11+30+45', '98/12/31 11*30*45' and '98@12@31 11^30^45' are equivalent.
- As a string in either 'YYYY-MM-DD' or 'YY-MM-DD' format. A "relaxed" syntax is allowed here, too. For example, '98-12-31', '98.12.31', '98/12/31' and '98@12@31' are equivalent.
- As a string with no delimiters in either 'YYYYMMDDHHMMSS' or 'YYMMDDHHMMSS' format, provided that the string makes sense as a date. For example, '19970523091528' and '970523091528' are interpreted as '1997-05-23 09:15:28', but '971122459015' is illegal (it has a nonsensical minute part) and becomes '0000-00-00 00:00:00'.
- As a string with no delimiters in either 'YYYYMMDD' or 'YYMMDD' format, provided that the string makes sense as a date. For example, '19970523' and '970523' are

interpreted as '1997-05-23', but '971332' is illegal (it has nonsensical month and day parts) and becomes '0000-00-00'.

- As a number in either YYYYMMDDHHMMSS or YYMMDDHHMMSS format, provided that the number makes sense as a date. For example, 19830905132800 and 830905132800 are interpreted as '1983-09-05 13:28:00'.
- As a number in either YYYYMMDD or YYMMDD format, provided that the number makes sense as a date. For example, 19830905 and 830905 are interpreted as '1983-09-05'.
- As the result of a function that returns a value that is acceptable in a DATETIME, DATE or TIMESTAMP context, such as NOW() or CURRENT_DATE.

Illegal DATETIME, DATE or TIMESTAMP values are converted to the "zero" value of the appropriate type ('0000-00-00 00:00:00', '0000-00-00' or 00000000000000).

For values specified as strings that include date part delimiters, it is not necessary to specify two digits for month or day values that are less than 10. '1979-6-9' is the same as '1979-06-09'. Similarly, for values specified as strings that include time part delimiters, it is not necessary to specify two digits for hour, month or second values that are less than 10. '1979-10-30 1:2:3' is the same as '1979-10-30 01:02:03'.

Values specified as numbers should be 6, 8, 12 or 14 digits long. If the number is 8 or 14 digits long, it is assumed to be in YYYYMMDD or YYYYMMDDHHMMSS format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in YYMMDD or YYMMDDHHMMSS format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as non-delimited strings are interpreted using their length as given. If the string is 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify '9903', thinking that will represent March, 1999, you will find that **MySQL** inserts a "zero" date into your table. This is because the year and month values are 99 and 03, but the day part is missing (zero), so the value is not a legal date.

TIMESTAMP columns store legal values using the full precision with which the value was specified, regardless of the display size. This has several implications:

- Always specify year, month, and day, even if your column types are TIMESTAMP(4) or TIMESTAMP(2). Otherwise, the value will not be a legal date and 0 will be stored.
- If you use ALTER TABLE to widen a narrow TIMESTAMP column, information will be displayed that previously was "hidden".
- Similarly, narrowing a TIMESTAMP column does not cause information to be lost, except in the sense that less information is shown when the values are displayed.
- Although TIMESTAMP values are stored to full precision, the only function that operates directly on the underlying stored value is UNIX_TIMESTAMP(). Other functions operate on the formatted retrieved value. This means you cannot use functions such as HOUR() or SECOND() unless the relevant part of the TIMESTAMP value is included in the formatted

value. For example, the HH part of a TIMESTAMP column is not displayed unless the display size is at least 10, so trying to use HOUR() on shorter TIMESTAMP values produces a meaningless result.

You can to some extent assign values of one date type to an object of a different date type. However, there may be some alteration of the value or loss of information:

- If you assign a DATE value to a DATETIME or TIMESTAMP object, the time part of the resulting value is set to '00:00:00', because the DATE value contains no time information.
- If you assign a DATETIME or TIMESTAMP value to a DATE object, the time part of the resulting value is deleted, because the DATE type stores no time information.
- Remember that although DATETIME, DATE and TIMESTAMP values all can be specified using the same set of formats, the types do not all have the same range of values. For example, TIMESTAMP values cannot be earlier than 1970 or later than 2037. This means that a date such as '1968-01-01', while legal as a DATETIME or DATE value, is not a valid TIMESTAMP value and will be converted to 0 if assigned to such an object.

Be aware of certain pitfalls when specifying date values:

- The relaxed format allowed for values specified as strings can be deceiving. For example, a value such as '10:11:12' might look like a time value because of the ':' delimiter, but if used in a date context will be interpreted as the year '2010-11-12'. The value '10:45:15' will be converted to '0000-00-00' because '45' is not a legal month.
- Year values specified as two digits are ambiguous, since the century is unknown. **MySQL** interprets 2-digit year values using the following rules:
  - Year values in the range 00-69 are converted to 2000-2069.
  - Year values in the range 70-99 are converted to 1970-1999.

7.2.3.3 The TIME type

MySQL retrieves and displays TIME values in 'HH:MM:SS' format (or 'HHH:MM:SS' format for large hours values). TIME values may range from '-838:59:59' to '838:59:59'. The reason the hours part may be so large is that the TIME type may be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

You can specify TIME values in a variety of formats:

- As a string in 'HH:MM:SS' format. A "relaxed" syntax is allowed—any non-numeric character may be used as the delimiter between time parts. For example, '10:11:12' and '10.11.12' are equivalent.
- As a string with no delimiters in 'HHMMSS' format, provided that it makes sense as a time. For example, '101112' is understood as '10:11:12', but '109712' is illegal (it has a nonsensical minute part) and becomes '00:00:00'.
- As a number in HHMMSS format, provided that it makes sense as a time. For example, 101112 is understood as '10:11:12'.
- As the result of a function that returns a value that is acceptable in a TIME context, such as CURRENT_TIME.

For TIME values specified as strings that include a time part delimiter, it is not necessary to specify two digits for hours, minutes or seconds values that are less than 10. '8:3:2' is the same as '08:03:02'.

Be careful about assigning "short" TIME values to a TIME column. **MySQL** interprets values using the assumption that the rightmost digits represent seconds. (**MySQL** interprets TIME values as elapsed time, rather than as time of day.) For example, you might think of '11:12', '1112' and 1112 as meaning '11:12:00' (12 minutes after 11 o'clock), but **MySQL** interprets them as '00:11:12' (11 minutes, 12 seconds). Similarly, '12' and 12 are interpreted as '00:00:12'.

Values that lie outside the TIME range but are otherwise legal are clipped to the appropriate endpoint of the range. For example, '-850:00:00' and '850:00:00' are converted to '-838:59:59' and '838:59:59'.

Illegal TIME values are converted to '00:00:00'. Note that since '00:00:00' is itself a legal TIME value, there is no way to tell, from a value of '00:00:00' stored in a table, whether the original value was specified as '00:00:00' or whether it was illegal.

7.2.3.4 The YEAR type

The YEAR type is a 1-byte type used for representing years.

MySQL retrieves and displays YEAR values in YYYY format. The range is 1901 to 2155.

You can specify YEAR values in a variety of formats:

- As a four-digit string in the range '1901' to '2155'.
- As a four-digit number in the range 1901 to 2155.
- As a two-digit string in the range '00' to '99'. Values in the ranges '00' to '69' and '70' to '99' are converted to YEAR values in the ranges 2000 to 2069 and 1970 to 1999.
- As a two-digit number in the range 1 to 99. Values in the ranges 1 to 69 and 70 to 99 are converted to YEAR values in the ranges 2001 to 2069 and 1970 to 1999. Note that the range for two-digit numbers is slightly different than the range for two-digit strings, since you cannot specify zero directly as a number and have it be interpreted as 2000. You *must* specify it as a string '0' or '00' or it will be interpreted as 0000.
- As the result of a function that returns a value that is acceptable in a YEAR context, such as NOW().

Illegal YEAR values are converted to 0000.

7.2.4 String types

The string types are CHAR, VARCHAR, BLOB, TEXT, ENUM and SET.

7.2.4.1 The CHAR and VARCHAR types

The CHAR and VARCHAR types are similar, but differ in the way they are stored and retrieved.

The length of a CHAR column is fixed to the length that you declare when you create the table. The length can be any value between 1 and 255. When CHAR values are stored,

they are right-padded with spaces to the specified length. When CHAR values are retrieved, trailing spaces are removed.

Values in VARCHAR columns are variable-length strings. You can declare a VARCHAR column to be any length between 1 and 255, just as for CHAR columns. However, in contrast to CHAR, VARCHAR values are stored using only as many characters as are needed, plus one byte to record the length. Values are not padded; instead, trailing spaces are removed when values are stored. (This space removal differs from the ANSI SQL specification.)

If you assign a value to a CHAR or VARCHAR column that exceeds the column's maximum length, the value is truncated to fit.

The table below illustrates the differences between the two types of columns by showing the result of storing various string values into CHAR(4) and VARCHAR(4) columns:

Value	CHAR(4)	Storage required	VARCHAR(4)	Storage required
''	' '	4 bytes	''	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

The values retrieved from the CHAR(4) and VARCHAR(4) columns will be the same in each case, because trailing spaces are removed from CHAR columns upon retrieval.

Values in CHAR and VARCHAR columns are sorted and compared in case-insensitive fashion, unless the BINARY attribute was specified when the table was created. The BINARY attribute means that column values are sorted and compared in case-sensitive fashion according to the ASCII order of the machine where the **MySQL** server is running.

The BINARY attribute is "sticky". This means that if a column marked BINARY is used in an expression, the whole expression is compared as a BINARY value.

MySQL may silently change the type of a CHAR or VARCHAR column at table creation time. See Section 7.6.1 [Silent column changes], page 172.

7.2.4.2 The BLOB and TEXT types

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types TINYBLOB, BLOB, MEDIUMBLOB and LONGBLOB differ only in the maximum length of the values they can hold. See Section 7.2.1 [Storage requirements], page 122.

The four TEXT types TINYTEXT, TEXT, MEDIUMTEXT and LONGTEXT correspond to the four BLOB types and have the same maximum lengths and storage requirements. The only difference between BLOB and TEXT types is that sorting and comparison is performed in case-sensitive fashion for BLOB values and case-insensitive fashion for TEXT values. In other words, a TEXT is a case-insensitive BLOB.

If you assign a value to a BLOB or TEXT column that exceeds the column type's maximum length, the value is truncated to fit.

In most respects, you can regard a TEXT column as a VARCHAR column that can be as big as you like. Similarly, you can regard a BLOB column as a VARCHAR BINARY column. The differences are:

- You can have indexes on BLOB and TEXT columns with **MySQL** versions 3.23.2 and newer. Older versions of **MySQL** did not support this.
- There is no trailing-space removal for BLOB and TEXT columns when values are stored, as there is for VARCHAR columns.
- BLOB and TEXT columns cannot have DEFAULT values.

MyODBC defines BLOB values as LONGVARBINARY and TEXT values as LONGVARCHAR.

Because BLOB and TEXT values may be extremely long, you may run up against some constraints when using them:

- If you want to use GROUP BY or ORDER BY on a BLOB or TEXT column, you must convert the column value into a fixed-length object. The standard way to do this is with the SUBSTRING function. For example:

```
mysql> select comment from tbl_name,substring(comment,20) as substr ORDER BY sub
```

  If you don't do this, only the first max_sort_length bytes of the column are used when sorting. The default value of max_sort_length is 1024; this value can be changed using the -O option when starting the mysqld server. You can group on an expression involving BLOB or TEXT values by specifying the column position or by using an alias:

```
mysql> select id,substring(blob_col,1,100) from tbl_name
           GROUP BY 2;
mysql> select id,substring(blob_col,1,100) as b from tbl_name
           GROUP BY b;
```

- The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you can actually transmit between the client and server is determined by the amount of available memory and the size of the communications buffers. You can change the message buffer size, but you must do so on both the server and client ends. See Section 10.1 [Server parameters], page 244.

Note that each BLOB or TEXT value is represented internally by a separately-allocated object. This is in contrast to all other column types, for which storage is allocated once per column when the table is opened.

7.2.4.3 The ENUM type

An ENUM is a string object whose value normally is chosen from a list of allowed values that are enumerated explicitly in the column specification at table creation time.

The value may also be the empty string ("") or NULL under certain circumstances:

- If you insert an invalid value into an ENUM (that is, a string not present in the list of allowed values), the empty string is inserted instead as a special error value.
- If an ENUM is declared NULL, NULL is also a legal value for the column, and the default value is NULL. If an ENUM is declared NOT NULL, the default value is the first element of the list of allowed values.

Each enumeration value has an index:

- Values from the list of allowable elements in the column specification are numbered beginning with 1.

- The index value of the empty string error value is 0. This means that you can use the following SELECT statement to find rows into which invalid ENUM values were assigned:

```
mysql> SELECT * FROM tbl_name WHERE enum_col=0;
```

- The index of the NULL value is NULL.

For example, a column specified as ENUM("one", "two", "three") can have any of the values shown below. The index of each value is also shown:

Value	Index
NULL	NULL
""	0
"one"	1
"two"	2
"three"	3

An enumeration can have a maximum of 65535 elements.

Lettercase is irrelevant when you assign values to an ENUM column. However, values retrieved from the column later have lettercase matching the values that were used to specify the allowable values at table creation time.

If you retrieve an ENUM in a numeric context, the column value's index is returned. If you store a number into an ENUM, the number is treated as an index, and the the value stored is the enumeration member with that index.

ENUM values are sorted according to the order in which the enumeration members were listed in the column specification. (In other words, ENUM values are sorted according to their index numbers.) For example, "a" sorts before "b" for ENUM("a", "b"), but "b" sorts before "a" for ENUM("b", "a"). The empty string sorts before non-empty strings, and NULL values sort before all other enumeration values.

If you want to get all possible values for an ENUM column, you should use: SHOW COLUMNS FROM table_name LIKE enum_column_name and parse the ENUM definition in the second column.

7.2.4.4 The SET type

A SET is a string object that can have zero or more values, each of which must be chosen from a list of allowed values specified when the table is created. SET column values that consist of multiple set members are specified with members separated by commas (','). A consequence of this is that SET member values cannot themselves contain commas.

For example, a column specified as SET("one", "two") NOT NULL can have any of these values:

```
""
"one"
"two"
"one,two"
```

A SET can have a maximum of 64 different members.

MySQL stores SET values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a SET value in a numeric context, the value

retrieved has bits set corresponding to the set members that make up the column value. If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. Suppose a column is specified as SET("a","b","c","d"). Then the members have the following bit values:

SET member	Decimal value	Binary value
a	1	0001
b	2	0010
c	4	0100
d	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members "a" and "d" are selected and the resulting value is "a,d".

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also doesn't not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value will appear once, with elements listed according to the order in which they were specified at table creation time. For example, if a column is specified as SET("a","b","c","d"), then "a,d", "d,a" and "d,a,a,d,d" will all appear as "a,d" when retrieved.

SET values are sorted numerically. NULL values sort before non-NULL SET values.

Normally, you perform a SELECT on a SET column using the LIKE operator or the FIND_IN_SET() function:

```
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
```

But the following will also work:

```
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
```

The first of these statements looks for an exact match. The second looks for values containing the first set member.

If you want to get all possible values for an SET column, you should use: SHOW COLUMNS FROM table_name LIKE set_column_name and parse the SET definition in the second column.

7.2.5 Choosing the right type for a column

For the most efficient use of storage, try to use the most precise type in all cases. For example, if an integer column will be used for values in the range between 1 and 99999, MEDIUMINT UNSIGNED is the best type.

Accurate representation of monetary values is a common problem. In **MySQL**, you should use the DECIMAL type. This is stored as a string, so no loss of accuracy should occur. If accuracy is not too important, the DOUBLE type may also be good enough.

For high precision, you can always convert to a fixed-point type stored in a BIGINT. This allows you to do all calculations with integers and convert results back to floating-point values only when necessary.

See Section 10.17 [Row format], page 260.

7.2.6 Column indexes

All **MySQL** column types can be indexed except `BLOB` and `TEXT` types. Use of indexes on the relevant columns is the best way to improve the performance of `SELECT` operations.

A table may have up to 16 indexes. The maximum index length is 256 bytes, although this may be changed when compiling **MySQL**.

You cannot index a column that may contain `NULL` values, so indexed columns must be declared `NOT NULL`.

For `CHAR` and `VARCHAR` columns, you can index a prefix of a column. This is much faster and requires less disk space than indexing the whole column. The syntax to use in the `CREATE TABLE` statement to index a column prefix looks like this:

```
KEY index_name (col_name(length))
```

The example below creates an index for the first 10 characters of the `name` column:

```
mysql> CREATE TABLE test (
           name CHAR(200) NOT NULL,
           KEY index_name (name(10)));
```

7.2.7 Multiple-column indexes

MySQL can create indexes on multiple columns. An index may consist of up to 15 columns. (On `CHAR` and `VARCHAR` columns you can also use a prefix of the column as a part of an index).

A multiple-column index can be considered a sorted array containing values that are created by concatenating the values of the indexed columns.

MySQL uses multiple-column indexes in such a way that queries are fast when you specify a known quantity for the first column of the index in a `WHERE` clause, even if you don't specify values for the other columns.

Suppose a table is created using the following specification:

```
mysql> CREATE TABLE test (
           id INT NOT NULL,
           last_name CHAR(30) NOT NULL,
           first_name CHAR(30) NOT NULL,
           PRIMARY KEY (id),
           INDEX name (last_name,first_name));
```

Then the index `name` is an index over `last_name` and `first_name`. The index will be used for queries that specify values in a known range for `last_name`, or for both `last_name` and `first_name`. Therefore, the `name` index will be used in the following queries:

```
mysql> SELECT * FROM test WHERE last_name="Widenius";

mysql> SELECT * FROM test WHERE last_name="Widenius"
                          AND first_name="Michael";

mysql> SELECT * FROM test WHERE last_name="Widenius"
                          AND (first_name="Michael" OR first_name="Monty");
```

```
mysql> SELECT * FROM test WHERE last_name="Widenius"
                           AND first_name >="M" AND first_name < "N";
```

However, the `name` index will NOT be used in the following queries:

```
mysql> SELECT * FROM test WHERE first_name="Michael";

mysql> SELECT * FROM test WHERE last_name="Widenius"
                           OR first_name="Michael";
```

For more information on the manner in which **MySQL** uses indexes to improve query performance, see Section 10.4 [**MySQL** indexes], page 249.

7.2.8 Using column types from other database engines

To make it easier to use code written for SQL implementations from other vendors, **MySQL** maps column types as shown in the table below. These mappings make it easier to move table definitions from other database engines to **MySQL**:

Other vendor type	MySQL type
`BINARY(NUM)`	`CHAR(NUM) BINARY`
`CHAR VARYING(NUM)`	`VARCHAR(NUM)`
`FLOAT4`	`FLOAT`
`FLOAT8`	`DOUBLE`
`INT1`	`TINYINT`
`INT2`	`SMALLINT`
`INT3`	`MEDIUMINT`
`INT4`	`INT`
`INT8`	`BIGINT`
`LONG VARBINARY`	`MEDIUMBLOB`
`LONG VARCHAR`	`MEDIUMTEXT`
`MIDDLEINT`	`MEDIUMINT`
`VARBINARY(NUM)`	`VARCHAR(NUM) BINARY`

Column type mapping occurs at table creation time. If you create a table with types used by other vendors and then issue a `DESCRIBE tbl_name` statement, **MySQL** reports the table structure using the equivalent **MySQL** types.

7.3 Functions for use in `SELECT` and `WHERE` clauses

A `select_expression` or `where_definition` in a SQL statement can consist of any expression using the functions described below.

An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression.

Note: There must be no whitespace between a function name and the parenthesis following it. This helps the **MySQL** parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. Spaces around arguments are permitted, though.

For the sake of brevity, examples display the output from the `mysql` program in abbreviated form. So this:

```
mysql> select MOD(29,9);
1 rows in set (0.00 sec)

+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
```

Is displayed like this:

```
mysql> select MOD(29,9);
        -> 2
```

7.3.1 Grouping functions

`( ... )` Parentheses. Use these to force the order of evaluation in an expression.

```
mysql> select 1+2*3;
        -> 7
mysql> select (1+2)*3;
        -> 9
```

7.3.2 Normal arithmetic operations

The usual arithmetic operators are available. Note that in the case of `-`, `+` and `*`, the result is calculated with `BIGINT` (64-bit) precision if both arguments are integers!

`+` Addition

```
mysql> select 3+5;
        -> 8
```

`-` Subtraction

```
mysql> select 3-5;
        -> -2
```

`*` Multiplication

```
mysql> select 3*5;
        -> 15
mysql> select 18014398509481984*18014398509481984.0;
        -> 324518553658426726783156020576256.0
mysql> select 18014398509481984*18014398509481984;
        -> 0
```

The result of the last expression is incorrect because the result of the integer multiplication exceeds the 64-bit range of `BIGINT` calculations.

`/` Division

```
mysql> select 3/5;
        -> 0.60
```

Division by zero produces a `NULL` result:

```
mysql> select 102/(1-1);
        -> NULL
```

A division will be calculated with `BIGINT` arithmetic only if performed in a context where its result is converted to an integer!

7.3.3 Bit functions

MySQL uses `BIGINT` (64-bit) arithmetic for bit operations, so these operators have a maximum range of 64 bits.

`|` Bitwise OR

```
mysql> select 29 | 15;
        -> 31
```

`&` Bitwise AND

```
mysql> select 29 & 15;
        -> 13
```

`<<` Shifts a longlong (`BIGINT`) number to the left.

```
mysql> select 1 << 2
        -> 4
```

`>>` Shifts a longlong (`BIGINT`) number to the right.

```
mysql> select 4 >> 2
        -> 1
```

`BIT_COUNT(N)`
Returns the number of bits that are set in the argument `N`.

```
mysql> select BIT_COUNT(29);
        -> 4
```

7.3.4 Logical operations

All logical functions return `1` (TRUE) or `0` (FALSE).

`NOT`
`!` Logical NOT. Returns `1` if the argument is `0`, otherwise returns `0`. Exception: `NOT NULL` returns `NULL`.

```
mysql> select NOT 1;
        -> 0
mysql> select NOT NULL;
        -> NULL
mysql> select ! (1+1);
        -> 0
```

```
mysql> select ! 1+1;
        -> 1
```

The last example returns `1` because the expression evaluates the same way as `(!1)+1`.

`OR`
`||` Logical OR. Returns `1` if either argument is not `0` and not `NULL`.

```
mysql> select 1 || 0;
        -> 1
mysql> select 0 || 0;
        -> 0
mysql> select 1 || NULL;
        -> 1
```

`AND`
`&&` Logical AND. Returns `0` if either argument is `0` or `NULL`, otherwise returns `1`.

```
mysql> select 1 && NULL;
        -> 0
mysql> select 1 && 0;
        -> 0
```

7.3.5 Comparison operators

Comparison operations result in a value of `1` (TRUE), `0` (FALSE) or `NULL`. These functions work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as needed (as in Perl).

MySQL performs comparisons using the following rules:

- If one or both arguments are `NULL`, the result of the comparison is `NULL`.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a `TIMESTAMP` or `DATETIME` column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly.
- In all other cases, the arguments are compared as floating-point (real) numbers.

By default, string comparisons are done in case-independent fashion using the current character set (ISO-8859-1 Latin1 by default, which also works excellently for English).

The examples below illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
```

```
mysql> SELECT 0 = 'x6';
         -> 1
```

`=` Equal

```
mysql> select 1 = 0;
        -> 0
mysql> select '0' = 0;
        -> 1
mysql> select '0.0' = 0;
        -> 1
mysql> select '0.01' = 0;
        -> 0
mysql> select '.01' = 0.01;
        -> 1
```

`<>`
`!=` Not equal

```
mysql> select '.01' <> '0.01';
        -> 1
mysql> select .01 <> '0.01';
        -> 0
mysql> select 'zapp' <> 'zappp';
        -> 1
```

`<=` Less than or equal

```
mysql> select 0.1 <= 2;
        -> 1
```

`<` Less than

```
mysql> select 2 <= 2;
        -> 1
```

`>=` Greater than or equal

```
mysql> select 2 >= 2;
        -> 1
```

`>` Greater than

```
mysql> select 2 > 2;
        -> 0
```

`<=>` Null safe equal

```
mysql> select 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1 1 0
```

`expr BETWEEN min AND max`

If `expr` is greater than or equal to `min` and `expr` is less than or equal to `max`, `BETWEEN` returns `1`, otherwise it returns `0`. This is equivalent to the expression `(min <= expr AND expr <= max)` if all the arguments are of the same type. The first argument (`expr`) determines how the comparison is performed. If `expr` is a string expression, a case-insensitive string comparison is done. If `expr` is a

binary string, a case-sensitive string comparison is done. If `expr` is an integer expression, an integer comparison is done. Otherwise, a floating-point (real) comparison is done.

```
mysql> select 1 BETWEEN 2 AND 3;
        -> 0
mysql> select 'b' BETWEEN 'a' AND 'c';
        -> 1
mysql> select 2 BETWEEN 2 AND '3';
        -> 1
mysql> select 2 BETWEEN 2 AND 'x-3';
        -> 0
```

`expr IN (value,...)`

Returns `1` if `expr` is any of the values in the `IN` list, else returns `0`. If all values are constants, then all values are evaluated according to the type of `expr` and sorted. The search for the item is then done using a binary search. This means `IN` is very quick if the `IN` value list consists entirely of constants. If `expr` is a case-sensitive string expression, the string comparison is performed in case-sensitive fashion.

```
mysql> select 2 IN (0,3,5,'wefwf');
        -> 0
mysql> select 'wefwf' IN (0,3,5,'wefwf');
        -> 1
```

`expr NOT IN (value,...)`

Same as `NOT (expr IN (value,...))`.

`ISNULL(expr)`

If `expr` is `NULL`, `ISNULL()` returns `1`, otherwise it returns `0`.

```
mysql> select ISNULL(1+1);
        -> 0
mysql> select ISNULL(1/0);
        -> 1
```

Note that a comparison of `NULL` values using `=` will always be false!

`COALESCE(list)`

Returns first non-`NULL` element in list.

```
mysql> select COALESCE(NULL,1);
        -> 1
mysql> select COALESCE(NULL,NULL,NULL);
        -> NULL
```

`INTERVAL(N,N1,N2,N3,...)`

Returns `0` if `N < N1`, `1` if `N < N2` and so on. All arguments are treated as numbers. It is required that `N1 < N2 < N3 < ... < Nn` for this function to work correctly. This is because a binary search is used (very fast).

```
mysql> select INTERVAL(23, 1, 15, 17, 30, 44, 200);
        -> 3
mysql> select INTERVAL(10, 1, 10, 100, 1000);
```

```
        -> 2
mysql> select INTERVAL(22, 23, 30, 44, 200);
        -> 0
```

7.3.6 String comparison functions

Normally, if any expression in a string comparison is case sensitive, the comparison is performed in case-sensitive fashion.

`expr LIKE pat [ESCAPE 'escape-char']`

Pattern matching using SQL simple regular expression comparison. Returns `1` (TRUE) or `0` (FALSE). With `LIKE` you can use the following two wildcard characters:

`%` Matches any number of characters, even zero characters

`_` Matches exactly one character

```
mysql> select 'David!' LIKE 'David_';
        -> 1
mysql> select 'David!' LIKE '%D%v%';
        -> 1
```

To test for literal instances of a wildcard character, precede the character with the escape character. If you don't specify the `ESCAPE` character, '\' is assumed:

`\%` Matches one `%` character

`\_` Matches one `_` character

```
mysql> select 'David!' LIKE 'David\_';
        -> 0
mysql> select 'David_' LIKE 'David\_';
        -> 1
```

To specify a different escape character, use the `ESCAPE` clause:

```
mysql> select 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

`LIKE` is allowed on numeric expressions! (This is a **MySQL** extension to the ANSI SQL `LIKE`.)

```
mysql> select 10 LIKE '1%';
        -> 1
```

Note: Because **MySQL** uses the C escape syntax in strings (e.g., '`\n`'), you must double any '\' that you use in your `LIKE` strings. For example, to search for '`\n`', specify it as '`\\n`'. To search for '\', specify it as '`\\\\`' (the backslashes are stripped once by the parser, and another time when the pattern match is done, leaving a single backslash to be matched).

`expr NOT LIKE pat [ESCAPE 'escape-char']`

Same as `NOT (expr LIKE pat [ESCAPE 'escape-char'])`.

`expr REGEXP pat`
`expr RLIKE pat`

Performs a pattern match of a string expression `expr` against a pattern `pat`. The pattern can be an extended regular expression. See Appendix H [Regexp],

page 450. Returns `1` if `expr` matches `pat`, otherwise returns `0`. `RLIKE` is a synonym for `REGEXP`, provided for `mSQL` compatibility. Note: Because **MySQL** uses the C escape syntax in strings (e.g., '`\n`'), you must double any '`\`' that you use in your `REGEXP` strings.

```
mysql> select 'Monty!' REGEXP 'm%y%%';
        -> 0
mysql> select 'Monty!' REGEXP '.*';
        -> 1
mysql> select 'new*\n*line' REGEXP 'new\\*.\\*line';
        -> 1
```

`REGEXP` and `RLIKE` use the current character set (ISO-8859-1 Latin1 by default) when deciding the type of a character.

`expr NOT REGEXP pat`
`expr NOT RLIKE pat`
: Same as `NOT (expr REGEXP pat)`.

`STRCMP(expr1,expr2)`
: `STRCMP()` returns `0` if the strings are the same, `-1` if the first argument is smaller than the second according to the current sort order, and `1` otherwise.

```
mysql> select STRCMP('text', 'text2');
        -> -1
mysql> select STRCMP('text2', 'text');
        -> 1
mysql> select STRCMP('text', 'text');
        -> 0
```

7.3.7 Cast operators

`BINARY`
: The `BINARY` operator casts the string following it to a binary string. This is an easy way to force a column comparison to be case independent even if the column isn't defined as `BINARY` or `BLOB`.

```
mysql> select "a" = "A";
        -> 1
mysql> select BINARY "a" = "A";
        -> 0
```

`BINARY` was introduced in **MySQL** 3.23.0

7.3.8 Control flow functions

`IFNULL(expr1,expr2)`
: If `expr1` is not `NULL`, `IFNULL()` returns `expr1`, else it returns `expr2`. `IFNULL()` returns a numeric or string value, depending on the context in which it is used.

```
mysql> select IFNULL(1,0);
        -> 1
mysql> select IFNULL(0,10);
        -> 0
mysql> select IFNULL(1/0,10);
        -> 10
mysql> select IFNULL(1/0,'yes');
        -> 'yes'
```

`IF(expr1,expr2,expr3)`

If `expr1` is TRUE (`expr1 <> 0` and `expr1 <> NULL`) then `IF()` returns `expr2`, else it returns `expr3`. `IFNULL()` returns a numeric or string value, depending on the context in which it is used.

```
mysql> select IF(1>2,2,3);
        -> 3
mysql> select IF(1<2,'yes','no');
        -> 'yes'
mysql> select IF(strcmp('test','test1'),'yes','no');
        -> 'no'
```

`expr1` is evaluated as an integer value, which means that if you are testing floating-point or string values, you should do so using a comparison operation.

```
mysql> select IF(0.1,1,0);
        -> 0
mysql> select IF(0.1<>0,1,0);
        -> 1
```

In the first case above, `IF(0.1)` returns `0` because `0.1` is converted to an integer value, resulting in a test of `IF(0)`. This may not be what you expect. In the second case, the comparison tests the original floating-point value to see whether it is non-zero. The result of the comparison is used as an integer.

`CASE value WHEN [compare-value] THEN result [WHEN [compare-value] THEN result ...] [ELSE result] END`

`CASE WHEN [condition] THEN result [WHEN [condition] THEN result ...] [ELSE result] END`

The first version returns the `result` where `value=compare-value`. The second version returns the result for the first condition which is true. If there was no matching result value, then the result after `ELSE` is returned. If there is no `ELSE` part then `NULL` is returned.

```
mysql> SELECT CASE 1 WHEN 1 THEN "one" WHEN 2 THEN "two" ELSE "more" END;
       -> "one"
mysql> SELECT CASE WHEN 1>0 THEN "true" ELSE "false" END;
       -> "true"
mysql> SELECT CASE BINARY "B" when "a" then 1 when "b" then 2 END;
       -> NULL
```

7.3.9 Mathematical functions

All mathematical functions return `NULL` in case of an error.

`-` Unary minus. Changes the sign of the argument.

```
mysql> select - 2;
        -> -2
```

Note that if this operator is used with a `BIGINT`, the return value is a `BIGINT`! This means that you should avoid using `-` on integers that may have the value of `-2^63`!

`ABS(X)` Returns the absolute value of `X`.

```
mysql> select ABS(2);
        -> 2
mysql> select ABS(-32);
        -> 32
```

This function is safe to use with `BIGINT` values.

`SIGN(X)` Returns the sign of the argument as `-1`, `0` or `1`, depending on whether `X` is negative, zero, or positive.

```
mysql> select SIGN(-32);
        -> -1
mysql> select SIGN(0);
        -> 0
mysql> select SIGN(234);
        -> 1
```

`MOD(N,M)`
`%` Modulo (like the `%` operator in C). Returns the remainder of `N` divided by `M`.

```
mysql> select MOD(234, 10);
        -> 4
mysql> select 253 % 7;
        -> 1
mysql> select MOD(29,9);
        -> 2
```

This function is safe to use with `BIGINT` values.

`FLOOR(X)` Returns the largest integer value not greater than `X`.

```
mysql> select FLOOR(1.23);
        -> 1
mysql> select FLOOR(-1.23);
        -> -2
```

Note that the return value is converted to a `BIGINT`!

`CEILING(X)`
Returns the smallest integer value not less than `X`.

```
mysql> select CEILING(1.23);
        -> 2
```

```
mysql> select CEILING(-1.23);
        -> -1
```

Note that the return value is converted to a `BIGINT`!

`ROUND(X)` Returns the argument `X`, rounded to an integer.

```
mysql> select ROUND(-1.23);
        -> -1
mysql> select ROUND(-1.58);
        -> -2
mysql> select ROUND(1.58);
        -> 2
```

Note that the return value is converted to a `BIGINT`!

`ROUND(X,D)`

Returns the argument `X`, rounded to a number with `D` decimals. If `D` is `0`, the result will have no decimal point or fractional part.

```
mysql> select ROUND(1.298, 1);
        -> 1.3
mysql> select ROUND(1.298, 0);
        -> 1
```

Note that the return value is converted to a `BIGINT`!

`EXP(X)` Returns the value of `e` (the base of natural logarithms) raised to the power of `X`.

```
mysql> select EXP(2);
        -> 7.389056
mysql> select EXP(-2);
        -> 0.135335
```

`LOG(X)` Returns the natural logarithm of `X`.

```
mysql> select LOG(2);
        -> 0.693147
mysql> select LOG(-2);
        -> NULL
```

If you want the log of a number `X` to some arbitary base `B`, use the formula `LOG(X)/LOG(B)`.

`LOG10(X)` Returns the base-10 logarithm of `X`.

```
mysql> select LOG10(2);
        -> 0.301030
mysql> select LOG10(100);
        -> 2.000000
mysql> select LOG10(-100);
        -> NULL
```

`POW(X,Y)`
`POWER(X,Y)`

Returns the value of `X` raised to the power of `Y`.

```
mysql> select POW(2,2);
        -> 4.000000
mysql> select POW(2,-2);
        -> 0.250000
```

`SQRT(X)` Returns the non-negative square root of `X`.

```
mysql> select SQRT(4);
        -> 2.000000
mysql> select SQRT(20);
        -> 4.472136
```

`PI()` Returns the value of PI.

```
mysql> select PI();
        -> 3.141593
```

`COS(X)` Returns the cosine of `X`, where `X` is given in radians.

```
mysql> select COS(PI());
        -> -1.000000
```

`SIN(X)` Returns the sine of `X`, where `X` is given in radians.

```
mysql> select SIN(PI());
        -> 0.000000
```

`TAN(X)` Returns the tangent of `X`, where `X` is given in radians.

```
mysql> select TAN(PI()+1);
        -> 1.557408
```

`ACOS(X)` Returns the arc cosine of `X`, that is, the value whose cosine is `X`. Returns `NULL` if `X` is not in the range `-1` to `1`.

```
mysql> select ACOS(1);
        -> 0.000000
mysql> select ACOS(1.0001);
        -> NULL
mysql> select ACOS(0);
        -> 1.570796
```

`ASIN(X)` Returns the arc sine of `X`, that is, the value whose sine is `X`. Returns `NULL` if `X` is not in the range `-1` to `1`.

```
mysql> select ASIN(0.2);
        -> 0.201358
mysql> select ASIN('foo');
        -> 0.000000
```

`ATAN(X)` Returns the arc tangent of `X`, that is, the value whose tangent is `X`.

```
mysql> select ATAN(2);
        -> 1.107149
mysql> select ATAN(-2);
        -> -1.107149
```

`ATAN2(X,Y)`

Returns the arc tangent of the two variables `X` and `Y`. It is similar to calculating the arc tangent of `Y / X`, except that the signs of both arguments are used to determine the quadrant of the result.

```
mysql> select ATAN(-2,2);
        -> -0.785398
mysql> select ATAN(PI(),0);
        -> 1.570796
```

`COT(X)` Returns the cotangent of `X`.

```
mysql> select COT(12);
        -> -1.57267341
mysql> select COT(0);
        -> NULL
```

`RAND()`

`RAND(N)` Returns a random floating-point value in the range `0` to `1.0`. If an integer argument `N` is specified, it is used as the seed value.

```
mysql> select RAND();
        -> 0.5925
mysql> select RAND(20);
        -> 0.1811
mysql> select RAND(20);
        -> 0.1811
mysql> select RAND();
        -> 0.2079
mysql> select RAND();
        -> 0.7888
```

You can't use a column with `RAND()` values in an `ORDER BY` clause, because `ORDER BY` would evaluate the column multiple times. In **MySQL** 3.23, you can however do: `SELECT * FROM table_name ORDER BY RAND()`

This is useful to get a random sample of a set `SELECT * FROM table1,table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000`.

`LEAST(X,Y,...)`

With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

- If the return value is used in an `INTEGER` context, or all arguments are integer-valued, they are compared as integers.
- If the return value is used in a `REAL` context, or all arguments are real-valued, they are compared as reals.
- If any argument is a case-sensitive string, the arguments are compared as case-sensitive strings.
- In other cases, the arguments are compared as case-insensitive strings.

```
mysql> select LEAST(2,0);
        -> 0
```

```
mysql> select LEAST(34.0,3.0,5.0,767.0);
        -> 3.0
mysql> select LEAST("B","A","C");
        -> "A"
```

In **MySQL** versions prior to 3.22.5, you can use `MIN()` instead of `LEAST`.

`GREATEST(X,Y,...)`

Returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for `LEAST`.

```
mysql> select GREATEST(2,0);
        -> 2
mysql> select GREATEST(34.0,3.0,5.0,767.0);
        -> 767.0
mysql> select GREATEST("B","A","C");
        -> "C"
```

In **MySQL** versions prior to 3.22.5, you can use `MAX()` instead of `GREATEST`.

`DEGREES(X)`

Returns the argument `X`, converted from radians to degrees.

```
mysql> select DEGREES(PI());
        -> 180.000000
```

`RADIANS(X)`

Returns the argument `X`, converted from degrees to radians.

```
mysql> select RADIANS(90);
        -> 1.570796
```

`TRUNCATE(X,D)`

Returns the number `X`, truncated to `D` decimals.

```
mysql> select TRUNCATE(1.223,1);
        -> 1.2
mysql> select TRUNCATE(1.999,1);
        -> 1.9
mysql> select TRUNCATE(1.999,0);
        -> 1
```

7.3.10 String functions

String-valued functions return `NULL` if the length of the result would be greater than the `max_allowed_packet` server parameter. See Section 10.1 [Server parameters], page 244.

For functions that operate on string positions, the first position is numbered 1.

`ASCII(str)`

Returns the ASCII code value of the leftmost character of the string `str`. Returns `0` if `str` is the empty string. Returns `NULL` if `str` is `NULL`.

```
mysql> select ASCII('2');
        -> 50
```

```
mysql> select ASCII(2);
        -> 50
mysql> select ASCII('dx');
        -> 100
```

`CONV(N,from_base,to_base)`
: Converts numbers between different number bases. Returns a string representation of the number `N`, converted from base `from_base` to base `to_base`. Returns `NULL` if any argument is `NULL`. The argument `N` is interpreted as an integer, but may be specified as an integer or a string. The minimum base is `2` and the maximum base is `36`. If `to_base` is a negative number, `N` is regarded as a signed number. Otherwise, `N` is treated as unsigned. `CONV` works with 64-bit precision.

```
mysql> select CONV("a",16,2);
        -> '1010'
mysql> select CONV("6E",18,8);
        -> '172'
mysql> select CONV(-17,10,-18);
        -> '-H'
mysql> select CONV(10+"10"+'10'+0xa,10,10);
        -> '40'
```

`BIN(N)`
: Returns a string representation of the binary value of `N`, where `N` is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,2)`. Returns `NULL` if `N` is `NULL`.

```
mysql> select BIN(12);
        -> '1100'
```

`OCT(N)`
: Returns a string representation of the octal value of `N`, where `N` is a longlong number. This is equivalent to `CONV(N,10,8)`. Returns `NULL` if `N` is `NULL`.

```
mysql> select OCT(12);
        -> '14'
```

`HEX(N)`
: Returns a string representation of the hexadecimal value of `N`, where `N` is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,16)`. Returns `NULL` if `N` is `NULL`.

```
mysql> select HEX(255);
        -> 'FF'
```

`CHAR(N,...)`
: `CHAR()` interprets the arguments as integers and returns a string consisting of the characters given by the ASCII code values of those integers. `NULL` values are skipped.

```
mysql> select CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> select CHAR(77,77.3,'77.3');
        -> 'MMM'
```

`CONCAT(X,Y,...)`

Returns the string that results from concatenating the arguments. Returns `NULL` if any argument is `NULL`. May have more than 2 arguments.

```
mysql> select CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> select CONCAT('My', NULL, 'QL');
        -> NULL
```

`LENGTH(str)`
`OCTET_LENGTH(str)`
`CHAR_LENGTH(str)`
`CHARACTER_LENGTH(str)`

Returns the length of the string `str`.

```
mysql> select LENGTH('text');
        -> 4
mysql> select OCTET_LENGTH('text');
        -> 4
```

`LOCATE(substr,str)`
`POSITION(substr IN str)`

Returns the position of the first occurrence of substring `substr` in string `str`. Returns `0` if `substr` is not in `str`.

```
mysql> select LOCATE('bar', 'foobarbar');
        -> 4
mysql> select LOCATE('xbar', 'foobar');
        -> 0
```

`LOCATE(substr,str,pos)`

Returns the position of the first occurrence of substring `substr` in string `str`, starting at position `pos`. Returns `0` if `substr` is not in `str`.

```
mysql> select LOCATE('bar', 'foobarbar',5);
        -> 7
```

`INSTR(str,substr)`

Returns the position of the first occurrence of substring `substr` in string `str`. This is the same as the two-argument form of `LOCATE()`, except that the arguments are swapped.

```
mysql> select INSTR('foobarbar', 'bar');
        -> 4
mysql> select INSTR('xbar', 'foobar');
        -> 0
```

`LPAD(str,len,padstr)`

Returns the string `str`, left-padded with the string `padstr` until `str` is `len` characters long.

```
mysql> select LPAD('hi',4,'??');
        -> '??hi'
```

`RPAD(str,len,padstr)`
: Returns the string `str`, right-padded with the string `padstr` until `str` is `len` characters long.

```
mysql> select RPAD('hi',5,'?');
        -> 'hi???'
```

`LEFT(str,len)`
: Returns the leftmost `len` characters from the string `str`.

```
mysql> select LEFT('foobarbar', 5);
        -> 'fooba'
```

`RIGHT(str,len)`
: Returns the rightmost `len` characters from the string `str`.

```
mysql> select RIGHT('foobarbar', 4);
        -> 'rbar'
mysql> select SUBSTRING('foobarbar' FROM 4);
        -> 'rbar'
```

`SUBSTRING(str,pos,len)`
`SUBSTRING(str FROM pos FOR len)`
`MID(str,pos,len)`
: Returns a substring `len` characters long from string `str`, starting at position `pos`. The variant form that uses `FROM` is ANSI SQL92 syntax.

```
mysql> select SUBSTRING('Quadratically',5,6);
        -> 'ratica'
```

`SUBSTRING(str,pos)`
`SUBSTRING(str FROM pos)`
: Returns a substring from string `str` starting at position `pos`.

```
mysql> select SUBSTRING('Quadratically',5);
        -> 'ratically'
```

`SUBSTRING_INDEX(str,delim,count)`
: Returns the substring from string `str` after `count` occurrences of the delimiter `delim`. If `count` is positive, everything to the left of the final delimiter (counting from the left) is returned. If `count` is negative, everything to the right of the final delimiter (counting from the right) is returned.

```
mysql> select SUBSTRING_INDEX('www.mysql.com', '.', 2);
        -> 'www.mysql'
mysql> select SUBSTRING_INDEX('www.mysql.com', '.', -2);
        -> 'mysql.com'
```

`LTRIM(str)`
: Returns the string `str` with leading space characters removed.

```
mysql> select LTRIM('  barbar');
        -> 'barbar'
```

`RTRIM(str)`
: Returns the string `str` with trailing space characters removed.

```
mysql> select RTRIM('barbar   ');
        -> 'barbar'
```

`TRIM([[BOTH | LEADING | TRAILING] [remstr] FROM] str)`
: Returns the string `str` with all `remstr` prefixes and/or suffixes removed. If none of the specifiers `BOTH`, `LEADING` or `TRAILING` are given, `BOTH` is assumed. If `remstr` is not specified, spaces are removed.

```
mysql> select TRIM('  bar   ');
        -> 'bar'
mysql> select TRIM(LEADING 'x' FROM 'xxxbarxxx');
        -> 'barxxx'
mysql> select TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -> 'bar'
mysql> select TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -> 'barx'
```

`SOUNDEX(str)`
: Returns a soundex string from `str`. Two strings that sound "about the same" should have identical soundex strings. A "standard" soundex string is 4 characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a "standard" soundex string. All non-alphanumeric characters are ignored in the given string. All international alpha characters outside the A-Z range are treated as vowels.

```
mysql> select SOUNDEX('Hello');
        -> 'H400'
mysql> select SOUNDEX('Quadratically');
        -> 'Q36324'
```

`SPACE(N)` Returns a string consisting of `N` space characters.

```
mysql> select SPACE(6);
        -> '      '
```

`REPLACE(str,from_str,to_str)`
: Returns the string `str` with all all occurrences of the string `from_str` replaced by the string `to_str`.

```
mysql> select REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

`REPEAT(str,count)`
: Returns a string consisting of the string `str` repeated `count` times. If `count <= 0`, returns an empty string. Returns `NULL` if `str` or `count` are `NULL`.

```
mysql> select REPEAT('MySQL', 3);
        -> 'MySQLMySQLMySQL'
```

`REVERSE(str)`
: Returns the string `str` with the order of the characters reversed.

```
mysql> select REVERSE('abc');
        -> 'cba'
```

`INSERT(str,pos,len,newstr)`
: Returns the string `str`, with the substring beginning at position `pos` and `len` characters long replaced by the string `newstr`.

```
mysql> select INSERT('Quadratic', 3, 4, 'What');
        -> 'QuWhattic'
```

`ELT(N,str1,str2,str3,...)`
: Returns `str1` if `N = 1`, `str2` if `N = 2`, and so on. Returns `NULL` if `N` is less than `1` or greater than the number of arguments. `ELT()` is the complement of `FIELD()`.

```
mysql> select ELT(1, 'ej', 'Heja', 'hej', 'foo');
        -> 'ej'
mysql> select ELT(4, 'ej', 'Heja', 'hej', 'foo');
        -> 'foo'
```

`FIELD(str,str1,str2,str3,...)`
: Returns the index of `str` in the `str1`, `str2`, `str3`, `...` list. Returns `0` if `str` is not found. `FIELD()` is the complement of `ELT()`.

```
mysql> select FIELD('ej', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 2
mysql> select FIELD('fo', 'Hej', 'ej', 'Heja', 'hej', 'foo');
        -> 0
```

`FIND_IN_SET(str,strlist)`
: Returns a value `1` to `N` if the string `str` is in the list `strlist` consisting of `N` substrings. A string list is a string composed of substrings separated by '`,`' characters. If the first argument is a constant string and the second is a column of type `SET`, the `FIND_IN_SET()` function is optimized to use bit arithmetic! Returns `0` if `str` is not in `strlist` or if `strlist` is the empty string. Returns `NULL` if either argument is `NULL`. This function will not work properly if the first argument contains a '`,`'.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

`MAKE_SET(bits,str1,str2,...)`
: Returns a set (a string containing substrings separated by '`,`' characters) consisting of the strings that have the corresponding bit in `bits` set. `str1` corresponds to bit 0, `str2` to bit 1, etc. `NULL` strings in `str1`, `str2`, `...` are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
        -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
        -> 'hello,world'
mysql> SELECT MAKE_SET(0,'a','b','c');
        -> ''
```

`EXPORT_SET(bits,on,off,[separator,[number_of_bits]])`
Returns a string where for every bit set in 'bit', you get a 'on' string and for every reset bit you get an 'off' string. Each string is separated with 'separator' (default ',') and only 'number_of_bits' (default 64) of 'bits' is used.

```
mysql> select EXPORT_SET(5,'Y','N',',',4)
        -> Y,N,Y,N
```

`LCASE(str)`
`LOWER(str)`
Returns the string `str` with all characters changed to lowercase according to the current character set mapping (the default is ISO-8859-1 Latin1).

```
mysql> select LCASE('QUADRATICALLY');
        -> 'quadratically'
```

`UCASE(str)`
`UPPER(str)`
Returns the string `str` with all characters changed to uppercase according to the current character set mapping (the default is ISO-8859-1 Latin1).

```
mysql> select UCASE('Hej');
        -> 'HEJ'
```

`LOAD_FILE(file_name)`
Reads the file and returns the file contents as a string. The file must be on the server, you must specify the full pathname to the file, and you must have the **file** privilege. The file must be readable by all and be smaller than `max_allowed_packet`.

If the file doesn't exist or can't be read due to one of the above reasons, the function returns `NULL`.

```
mysql> UPDATE table_name SET blob_column=LOAD_FILE("/tmp/picture") WHERE
```

There is no string function to convert a number to a char. There is no need for one, because **MySQL** automatically converts numbers to strings as necessary, and vice versa:

```
mysql> SELECT 1+"1";
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This only affects comparisons.

7.3.11 Date and time functions

See Section 7.2.3 [Date and time types], page 125 for a description of the range of values each type has, and the valid formats in which date and time values may be specified.

Here is an example that uses date functions. The query below selects all records with a `date_col` value from within the last 30 days:

```
mysql> SELECT something FROM table
           WHERE TO_DAYS(NOW()) - TO_DAYS(date_col) <= 30;
```

`DAYOFWEEK(date)`
: Returns the weekday index for `date` (`1` = Sunday, `2` = Monday, ... `7` = Saturday). These index values correspond to the ODBC standard.

```
mysql> select DAYOFWEEK('1998-02-03');
        -> 3
```

`WEEKDAY(date)`
: Returns the weekday index for `date` (`0` = Monday, `1` = Tuesday, ... `6` = Sunday).

```
mysql> select WEEKDAY('1997-10-04 22:23:00');
        -> 5
mysql> select WEEKDAY('1997-11-05');
        -> 2
```

`DAYOFMONTH(date)`
: Returns the day of the month for `date`, in the range `1` to `31`.

```
mysql> select DAYOFMONTH('1998-02-03');
        -> 3
```

`DAYOFYEAR(date)`
: Returns the day of the year for `date`, in the range `1` to `366`.

```
mysql> select DAYOFYEAR('1998-02-03');
        -> 34
```

`MONTH(date)`
: Returns the month for `date`, in the range `1` to `12`.

```
mysql> select MONTH('1998-02-03');
        -> 2
```

`DAYNAME(date)`
: Returns the name of the weekday for `date`.

```
mysql> select DAYNAME("1998-02-05");
        -> 'Thursday'
```

`MONTHNAME(date)`
: Returns the name of the month for `date`.

```
mysql> select MONTHNAME("1998-02-05");
        -> 'February'
```

`QUARTER(date)`
: Returns the quarter of the year for `date`, in the range `1` to `4`.

```
mysql> select QUARTER('98-04-01');
        -> 2
```

`WEEK(date)`
`WEEK(date,first)`
: With a single argument, returns the week for `date`, in the range `0` to `52`, for locations where Sunday is the first day of the week. The two-argument form of

`WEEK()` allows you to specify whether the week starts on Sunday or Monday. The week starts on Sunday if the second argument is `0`, on Monday if the second argument is `1`.

```
mysql> select WEEK('1998-02-20');
        -> 7
mysql> select WEEK('1998-02-20',0);
        -> 7
mysql> select WEEK('1998-02-20',1);
        -> 8
```

`YEAR(date)`
Returns the year for `date`, in the range `1000` to `9999`.

```
mysql> select YEAR('98-02-03');
        -> 1998
```

`HOUR(time)`
Returns the hour for `time`, in the range `0` to `23`.

```
mysql> select HOUR('10:05:03');
        -> 10
```

`MINUTE(time)`
Returns the minute for `time`, in the range `0` to `59`.

```
mysql> select MINUTE('98-02-03 10:05:03');
        -> 5
```

`SECOND(time)`
Returns the second for `time`, in the range `0` to `59`.

```
mysql> select SECOND('10:05:03');
        -> 3
```

`PERIOD_ADD(P,N)`
Adds `N` months to period `P` (in the format `YYMM` or `YYYYMM`). Returns a value in the format `YYYYMM`.

Note that the period argument `P` is *not* a date value.

```
mysql> select PERIOD_ADD(9801,2);
        -> 199803
```

`PERIOD_DIFF(P1,P2)`
Returns the number of months between periods `P1` and `P2`. `P1` and `P2` should be in the format `YYMM` or `YYYYMM`.

Note that the period arguments `P1` and `P2` are *not* date values.

```
mysql> select PERIOD_DIFF(9802,199703);
        -> 11
```

`DATE_ADD(date,INTERVAL expr type)`
`DATE_SUB(date,INTERVAL expr type)`
`ADDDATE(date,INTERVAL expr type)`
`SUBDATE(date,INTERVAL expr type)`
These functions perform date arithmetic. They are new for **MySQL** 3.22. `ADDDATE()` and `SUBDATE()` are synonyms for `DATE_ADD()` and `DATE_SUB()`.

`date` is a `DATETIME` or `DATE` value specifying the starting date. `expr` is an expression specifying the interval value to be added or substracted from the starting date. `expr` is a string; it may start with a '`-`' for negative intervals. `type` is a keyword indicating how the expression should be interpreted.

The `EXTRACT()` function returns the corresponding interval from the date.

The following table shows how the `type` and `expr` arguments are related:

`type` **value**	**Meaning**	**Expected** `expr` **format**
`SECOND`	Seconds	`SECONDS`
`MINUTE`	Minutes	`MINUTES`
`HOUR`	Hours	`HOURS`
`DAY`	Days	`DAYS`
`MONTH`	Months	`MONTHS`
`YEAR`	Years	`YEARS`
`MINUTE_SECOND`	Minutes and seconds	`"MINUTES:SECONDS"`
`HOUR_MINUTE`	Hours and minutes	`"HOURS:MINUTES"`
`DAY_HOUR`	Days and hours	`"DAYS HOURS"`
`YEAR_MONTH`	Years and months	`"YEARS-MONTHS"`
`HOUR_SECOND`	Hours, minutes,	`"HOURS:MINUTES:SECONDS"`
`DAY_MINUTE`	Days, hours, minutes	`"DAYS HOURS:MINUTES"`
`DAY_SECOND`	Days, hours, minutes, seconds	`"DAYS HOURS:MINUTES:SECONDS"`

MySQL allows any non-numeric delimiter in the `expr` format. The ones shown in the table are the suggested delimiters. If the `date` argument is a `DATE` value and your calculations involve only `YEAR`, `MONTH` and `DAY` parts (that is, no time parts), the result is a `DATE` value. Otherwise the result is a `DATETIME` value.

```
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL 1 SECOND);
        -> 1998-01-01 00:00:00
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL 1 DAY);
        -> 1998-01-01 23:59:59
mysql> SELECT DATE_ADD("1997-12-31 23:59:59",
                       INTERVAL "1:1" MINUTE_SECOND);
        -> 1998-01-01 00:01:00
mysql> SELECT DATE_SUB("1998-01-01 00:00:00",
                       INTERVAL "1 1:1:1" DAY_SECOND);
        -> 1997-12-30 22:58:59
mysql> SELECT DATE_ADD("1998-01-01 00:00:00",
                       INTERVAL "-1 10" DAY_HOUR);
        -> 1997-12-30 14:00:00
mysql> SELECT DATE_SUB("1998-01-02", INTERVAL 31 DAY);
        -> 1997-12-02
mysql> SELECT EXTRACT(YEAR FROM "1999-07-02");
       -> 1999
mysql> SELECT EXTRACT(YEAR_MONTH FROM "1999-07-02 01:02:03");
       -> 199907
```

```
mysql> SELECT EXTRACT(DAY_MINUTE FROM "1999-07-02 01:02:03");
       -> 20102
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the `type` keyword), **MySQL** assumes you have left out the leftmost parts of the interval value. For example, if you specify a `type` of `DAY_SECOND`, the value of `expr` is expected to have days, hours, minutes and seconds parts. If you specify a value like `"1:10"`, **MySQL** assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `"1:10" DAY_SECOND` is interpreted in such a way that it is equivalent to `"1:10" MINUTE_SECOND`. This is analogous to the way that **MySQL** interprets `TIME` values as representing elapsed time rather than as time of day.

If you use incorrect dates, the result is `NULL`. If you add `MONTH`, `YEAR_MONTH` or `YEAR` and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month.

```
mysql> select DATE_ADD('1998-01-30', Interval 1 month);
        -> 1998-02-28
```

Note from the preceding example that the word `INTERVAL` and the `type` keyword are not case sensitive.

`TO_DAYS(date)`

Given a date `date`, returns a daynumber (the number of days since year 0).

```
mysql> select TO_DAYS(950501);
        -> 728779
mysql> select TO_DAYS('1997-10-07');
        -> 729669
```

`TO_DAYS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582).

`FROM_DAYS(N)`

Given a daynumber `N`, returns a `DATE` value.

```
mysql> select FROM_DAYS(729669);
        -> '1997-10-07'
```

`FROM_DAYS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582).

`DATE_FORMAT(date,format)`

Formats the `date` value according to the `format` string. The following specifiers may be used in the `format` string:

`%M`	Month name (`January`..`December`)
`%W`	Weekday name (`Sunday`..`Saturday`)
`%D`	Day of the month with english suffix (`1st`, `2nd`, `3rd`, etc.)
`%Y`	Year, numeric, 4 digits
`%y`	Year, numeric, 2 digits
`%a`	Abbreviated weekday name (`Sun`..`Sat`)
`%d`	Day of the month, numeric (`00`..`31`)

%e	Day of the month, numeric (0..31)
%m	Month, numeric (01..12)
%c	Month, numeric (1..12)
%b	Abbreviated month name (Jan..Dec)
%j	Day of year (001..366)
%H	Hour (00..23)
%k	Hour (0..23)
%h	Hour (01..12)
%I	Hour (01..12)
%l	Hour (1..12)
%i	Minutes, numeric (00..59)
%r	Time, 12-hour (hh:mm:ss [AP]M)
%T	Time, 24-hour (hh:mm:ss)
%S	Seconds (00..59)
%s	Seconds (00..59)
%p	AM or PM
%w	Day of the week (0=Sunday..6=Saturday)
%U	Week (0..52), where Sunday is the first day of the week
%u	Week (0..52), where Monday is the first day of the week
%%	A literal '%'.

All other characters are just copied to the result without interpretation.

```
mysql> select DATE_FORMAT('1997-10-04 22:23:00', '%W %M %Y');
        -> 'Saturday October 1997'
mysql> select DATE_FORMAT('1997-10-04 22:23:00', '%H:%i:%s');
        -> '22:23:00'
mysql> select DATE_FORMAT('1997-10-04 22:23:00',
                          '%D %y %a %d %m %b %j');
        -> '4th 97 Sat 04 10 Oct 277'
mysql> select DATE_FORMAT('1997-10-04 22:23:00',
                          '%H %k %I %r %T %S %w');
        -> '22 22 10 10:23:00 PM 22:23:00 00 6'
```

As of **MySQL** 3.23, the % is required before a format specifier characters. In earlier versions of **MySQL**, % was optional.

`TIME_FORMAT(time,format)`

This is used like the `DATE_FORMAT()` function above, but the `format` string may contain only those format specifiers that handle hours, minutes and seconds. Other specifiers produce a `NULL` value or `0`.

`CURDATE()`
`CURRENT_DATE`

Returns today's date as a value in `'YYYY-MM-DD'` or `YYYYMMDD` format, depending on whether the function is used in a string or numeric context.

```
mysql> select CURDATE();
        -> '1997-12-15'
mysql> select CURDATE() + 0;
        -> 19971215
```

CURTIME()
CURRENT_TIME
: Returns the current time as a value in 'HH:MM:SS' or HHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> select CURTIME();
        -> '23:50:26'
mysql> select CURTIME() + 0;
        -> 235026
```

NOW()
SYSDATE()
CURRENT_TIMESTAMP
: Returns the current date and time as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> select NOW();
        -> '1997-12-15 23:50:26'
mysql> select NOW() + 0;
        -> 19971215235026
```

UNIX_TIMESTAMP()
UNIX_TIMESTAMP(date)
: If called with no argument, returns a Unix timestamp (seconds since '1970-01-01 00:00:00' GMT). If UNIX_TIMESTAMP() is called with a date argument, it returns the value of the argument as seconds since '1970-01-01 00:00:00' GMT. date may be a DATE string, a DATETIME string, a TIMESTAMP, or a number in the format YYMMDD or YYYYMMDD in local time.

```
mysql> select UNIX_TIMESTAMP();
        -> 882226357
mysql> select UNIX_TIMESTAMP('1997-10-04 22:23:00');
        -> 875996580
```

When UNIX_TIMESTAMP is used on a TIMESTAMP column, the function will receive the value directly, with no implicit "string-to-unix-timestamp" conversion.

FROM_UNIXTIME(unix_timestamp)
: Returns a representation of the unix_timestamp argument as a value in 'YYYY-MM-DD HH:MM:SS' or YYYYMMDDHHMMSS format, depending on whether the function is used in a string or numeric context.

```
mysql> select FROM_UNIXTIME(875996580);
        -> '1997-10-04 22:23:00'
mysql> select FROM_UNIXTIME(875996580) + 0;
        -> 19971004222300
```

FROM_UNIXTIME(unix_timestamp,format)
: Returns a string representation of the Unix timestamp, formatted according to the format string. format may contain the same specifiers as those listed in the entry for the DATE_FORMAT() function.

```
mysql> select FROM_UNIXTIME(UNIX_TIMESTAMP(),
```

```
                                          '%Y %D %M %h:%i:%s %x');
                    -> '1997 23rd December 03:43:30 x'
```

`SEC_TO_TIME(seconds)`
: Returns the `seconds` argument, converted to hours, minutes and seconds, as a value in `'HH:MM:SS'` or `HHMMSS` format, depending on whether the function is used in a string or numeric context.

```
mysql> select SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> select SEC_TO_TIME(2378) + 0;
        -> 3938
```

`TIME_TO_SEC(time)`
: Returns the `time` argument, converted to seconds.

```
mysql> select TIME_TO_SEC('22:23:00');
        -> 80580
mysql> select TIME_TO_SEC('00:39:38');
        -> 2378
```

7.3.12 Miscellaneous functions

`DATABASE()`
: Returns the current database name.

```
mysql> select DATABASE();
        -> 'test'
```

If there is no current database, `DATABASE()` returns the empty string.

`USER()`
`SYSTEM_USER()`
`SESSION_USER()`
: Returns the current **MySQL** user name.

```
mysql> select USER();
        -> 'davida@localhost'
```

In MySQL 3.22.11 or later, this includes the client hostname as well as the username. You can extract just the username part like this (which works whether or not the value includes a hostname part):

```
mysql> select substring_index(USER(),"@",1);
        -> 'davida'
```

`PASSWORD(str)`
: Calculates a password string from the plaintext password `str`. This is the function that is used for encrypting **MySQL** passwords for storage in the `Password` column of the `user` grant table.

```
mysql> select PASSWORD('badpwd');
        -> '7f84554057dd964b'
```

`PASSWORD()` encryption is non-reversible.

PASSWORD() does not perform password encryption in the same way that Unix passwords are encrypted. You should not assume that if your Unix password and your **MySQL** password are the same, PASSWORD() will result in the same encrypted value as is stored in the Unix password file. See ENCRYPT().

ENCRYPT(str[,salt])
: Encrypt str using the Unix crypt() system call. The salt argument should be a string with 2 characters.

```
mysql> select ENCRYPT("hello");
        -> 'VxuFAJXVARROc'
```

If crypt() is not available on your system, ENCRYPT() always returns NULL.

ENCRYPT() ignores all but the first 8 characters of str, at least on some systems. This will be determined by the behavior of the underlying crypt() system call.

ENCODE(str,pass_str)
: Encrypt str using pass_str as the password. To decrypt the result, use DECODE().

The results is a binary string. If you want to save it in a column, use a BLOB column type.

DECODE(crypt_str,pass_str)
: Descrypts the encrypted string crypt_str using pass_str as the password. crypt_str should be a string returned from ENCODE().

MD5(string)
: Calculates a MD5 checksum for the string. Value is returned as a 32 long hex number that may, for example, be used as a hash key.

```
mysql> select MD5("testing")
        -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

This is a "RSA Data Security, Inc. MD5 Message-Digest Algorithm".

LAST_INSERT_ID([expr])
: Returns the last automatically generated value that was inserted into an AUTO_INCREMENT column. See Section 20.4.29 [mysql_insert_id(), page 348.

```
mysql> select LAST_INSERT_ID();
        -> 195
```

The last ID that was generated is maintained in the server on a per-connection basis. It will not be changed by another client. It will not even be changed if you update another AUTO_INCREMENT column with a non-magic value (that is, a value that is not NULL and not 0).

If expr is given as an argument to LAST_INSERT_ID() in an UPDATE clause, then the value of the argument is returned as a LAST_INSERT_ID() value. This can be used to simulate sequences:

First create the table:

```
mysql> create table sequence (id int not null);
mysql> insert into sequence values (0);
```

Then the table can be used to generate sequence numbers like this:

```
mysql> update sequence set id=LAST_INSERT_ID(id+1);
```

You can generate sequences without calling `LAST_INSERT_ID()`, but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. You can retrieve the new ID as you would read any normal `AUTO_INCREMENT` value in **MySQL**. For example, `LAST_INSERT_ID()` (without an argument) will return the new ID. The C API function `mysql_insert_id()` can also be used to get the value.

`FORMAT(X,D)`

Formats the number `X` to a format like `'#,###,###.##'` with `D` decimals. If `D` is `0`, the result will have no decimal point or fractional part.

```
mysql> select FORMAT(12332.1234, 2);
        -> '12,332.12'
mysql> select FORMAT(12332.1,4);
        -> '12,332.1000'
mysql> select FORMAT(12332.2,0);
        -> '12,332'
```

`VERSION()`

Returns a string indicating the **MySQL** server version.

```
mysql> select VERSION();
        -> '3.22.19b-log'
```

`GET_LOCK(str,timeout)`

Tries to obtain a lock with a name given by the string `str`, with a timeout of `timeout` seconds. Returns `1` if the lock was obtained successfully, `0` if the attempt timed out, or `NULL` if an error occurred (such as running out of memory or the thread was killed with `mysqladmin kill`). A lock is released when you execute `RELEASE_LOCK()`, execute a new `GET_LOCK()` or the thread terminates. This function can be used to implement application locks or to simulate record locks.

```
mysql> select GET_LOCK("lock1",10);
        -> 1
mysql> select GET_LOCK("lock2",10);
        -> 1
mysql> select RELEASE_LOCK("lock2");
        -> 1
mysql> select RELEASE_LOCK("lock1");
        -> NULL
```

Note that the second `RELEASE_LOCK()` call returns `NULL` because the lock `"lock1"` was automatically released by the second `GET_LOCK()` call.

`RELEASE_LOCK(str)`

Releases the lock named by the string `str` that was obtained with `GET_LOCK()`. Returns `1` if the lock was released, `0` if the lock wasn't locked by this thread (in which case the lock is not released) and `NULL` if the named lock didn't exist. The lock will not exist if it was never obtained by a call to `GET_LOCK()` or if it already has been released.

`BENCHMARK(count,expr)`

The `BENCHMARK()` function executes the expression `expr` repeatedly `count` times. It may be used to time how fast **MySQL** processes the expression. The result value is always `0`. The intended use is in the `mysql` client, which reports query execution times.

```
mysql> select BENCHMARK(1000000,encode("hello","goodbye"));
+-------------------------------------------+
| BENCHMARK(1000000,encode("hello","goodbye")) |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
1 row in set (4.74 sec)
```

The time reported is elapsed time on the client end, not CPU time on the server end. It may be advisable to execute `BENCHMARK()` several times, and interpret the result with regard to how heavily loaded the server machine is.

7.3.13 Functions for use with `GROUP BY` clauses

If you use a group function in a statement containing no `GROUP BY` clause, it is equivalent to grouping on all rows.

`COUNT(expr)`

Returns a count of the number of non-`NULL` rows retrieved by a `SELECT` statement.

```
mysql> select student.student_name,COUNT(*)
           from student,course
           where student.student_id=course.student_id
           GROUP BY student_name;
```

`COUNT(*)` is optimized to return very quickly if the `SELECT` retrieves from one table, no other columns are retrieved and there is no `WHERE` clause. For example:

```
mysql> select COUNT(*) from student;
```

`COUNT(DISTINCT expr,[expr...])`

Returns a count of the number of different values.

```
mysql> select COUNT(DISTINCT results) from student;
```

In **MySQL** you can get the number of distinct expressions combinations by giving a list of expressions. In ANSI SQL you would have to do a concatenation of all expressions inside `CODE(DISTINCT ..)`.

`AVG(expr)`

Returns the average value of `expr`.

```
mysql> select student_name, AVG(test_score)
           from student
           GROUP BY student_name;
```

`MIN(expr)`
`MAX(expr)`
: Returns the minimum or maximum value of `expr`. `MIN()` and `MAX()` may take a string argument; in such cases they return the minimum or maximum string value.

```
mysql> select student_name, MIN(test_score), MAX(test_score)
           from student
           GROUP BY student_name;
```

`SUM(expr)`
: Returns the sum of `expr`. Note that if the return set has no rows, it returns NULL!

`STD(expr)`
`STDDEV(expr)`
: Returns the standard deviation of `expr`. This is an extension to ANSI SQL. The `STDDEV()` form of this function is provided for Oracle compatability.

`BIT_OR(expr)`
: Returns the bitwise `OR` of all bits in `expr`. The calculation is performed with 64-bit (`BIGINT` precision.

`BIT_AND(expr)`
: Returns the bitwise `AND` of all bits in `expr`. The calculation is performed with 64-bit (`BIGINT` precision.

MySQL has extended the use of `GROUP BY`. You can use columns or calculations in the `SELECT` expressions which don't appear in the `GROUP BY` part. This stands for *any possible value for this group*. You can use this to get better performance by avoiding sorting and grouping on unnecessary items. For example, you don't need to group on `customer.name` in the following query:

```
mysql> select order.custid,customer.name,max(payments)
       from order,customer
       where order.custid = customer.custid
       GROUP BY order.custid;
```

In ANSI SQL, you would have to add `customer.name` to the `GROUP BY` clause. In **MySQL**, the name is redundant.

Don't use this feature if the columns you omit from the `GROUP BY` part aren't unique in the group!

In some cases, you can use `MIN()` and `MAX()` to obtain a specific column value even if it isn't unique. The following gives the value of `column` from the row containing the smallest value in the `sort` column:

```
substr(MIN(concat(sort,space(6-length(sort)),column),7,length(column)))
```

Note that if you are using **MySQL** 3.22 (or earlier) or if you are trying to follow ANSI SQL, you can't use expressions in `GROUP BY` or `ORDER BY` clauses. You can work around this limitation by using an alias for the expression:

```
mysql> SELECT id,FLOOR(value/100) AS val FROM tbl_name
```

```
                GROUP BY id,val ORDER BY val;
```

In MySQL 3.23 you can do:

```
mysql> SELECT id,FLOOR(value/100) FROM tbl_name ORDER BY RAND();
```

7.4 CREATE DATABASE syntax

```
CREATE DATABASE db_name
```

CREATE DATABASE creates a database with the given name. Rules for allowable database names are given in Section 7.1.5 [Legal names], page 118. An error occurs if the database already exists.

Databases in **MySQL** are implemented as directories containing files that correspond to tables in the database. Since there are no tables in a database when it is initially created, the CREATE DATABASE statement only creates a directory under the **MySQL** data directory.

You can also create databases with mysqladmin. See Section 12.1 [Programs], page 266.

7.5 DROP DATABASE syntax

```
DROP DATABASE [IF EXISTS] db_name
```

DROP DATABASE drops all tables in the database and deletes the database. **Be VERY careful with this command!**

DROP DATABASE returns the number of files that were removed from the database directory. Normally, this is three times the number of tables, since each table corresponds to a '.ISD' file, a '.ISM' file and a '.frm' file.

In **MySQL** 3.22 or later, you can use the keywords IF EXISTS to prevent an error from occurring if the database doesn't exist.

You can also drop databases with mysqladmin. See Section 12.1 [Programs], page 266.

7.6 CREATE TABLE syntax

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name (create_definition,...)
[table_options] [select_statement]

create_definition:
  col_name type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
              [PRIMARY KEY] [reference_definition]
  or    PRIMARY KEY (index_col_name,...)
  or    KEY [index_name] KEY(index_col_name,...)
  or    INDEX [index_name] (index_col_name,...)
  or    UNIQUE [INDEX] [index_name] (index_col_name,...)
  or    [CONSTRAINT symbol] FOREIGN KEY index_name (index_col_name,...)
              [reference_definition]
  or    CHECK (expr)

type:
        TINYINT[(length)] [UNSIGNED] [ZEROFILL]
```

```
  or    SMALLINT[(length)] [UNSIGNED] [ZEROFILL]
  or    MEDIUMINT[(length)] [UNSIGNED] [ZEROFILL]
  or    INT[(length)] [UNSIGNED] [ZEROFILL]
  or    INTEGER[(length)] [UNSIGNED] [ZEROFILL]
  or    BIGINT[(length)] [UNSIGNED] [ZEROFILL]
  or    REAL[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    DOUBLE[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    FLOAT[(length,decimals)] [UNSIGNED] [ZEROFILL]
  or    DECIMAL(length,decimals) [UNSIGNED] [ZEROFILL]
  or    NUMERIC(length,decimals) [UNSIGNED] [ZEROFILL]
  or    CHAR(length) [BINARY]
  or    VARCHAR(length) [BINARY]
  or    DATE
  or    TIME
  or    TIMESTAMP
  or    DATETIME
  or    TINYBLOB
  or    BLOB
  or    MEDIUMBLOB
  or    LONGBLOB
  or    TINYTEXT
  or    TEXT
  or    MEDIUMTEXT
  or    LONGTEXT
  or    ENUM(value1,value2,value3,...)
  or    SET(value1,value2,value3,...)

index_col_name:
        col_name [(length)]

reference_definition:
        REFERENCES tbl_name [(index_col_name,...)]
                   [MATCH FULL | MATCH PARTIAL]
                   [ON DELETE reference_option]
                   [ON UPDATE reference_option]

reference_option:
        RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:
type = [ISAM | MYISAM | HEAP]
or AUTO_INCREMENT = #
or AVG_ROW_LENGTH = #
or CHECKSUM = [0 | 1]
or COMMENT = "string"
or MAX_ROWS = #
or MIN_ROWS = #
or PACK_KEYS = [0 | 1]
```

```
or PASSWORD = "string"
or DELAY_KEY_WRITE = [0 | 1]

select_statement:
[IGNORE | REPLACE] SELECT ...  (Some legal select statement)
```

`CREATE TABLE` creates a table with the given name in the current database. Rules for allowable table names are given in Section 7.1.5 [Legal names], page 118. An error occurs if there is no current database or if the table already exists.

In **MySQL** 3.22 or later, the table name can be specified as `db_name.tbl_name`. This works whether or not there is a current database.

In **MySQL** 3.23, you can use the `TEMPORARY` keyword when you create a table. A temporary table will automatically be deleted if a connection dies and the name is per connection. This means that two different connections can both use the same temporary table name without conflicting with each other or with an existing table of the same name. (The existing table is hidden until the temporary table is deleted).

In **MySQL** 3.23 or later, you can use the keywords `IF NOT EXISTS` so that an error does not occur if the table already exists. Note that there is no verification that the table structures are identical.

Each table `tbl_name` is represented by some files in the database directory. In the case of ISAM-type tables you will get:

File	Purpose
`tbl_name.frm`	Table definition (form) file
`tbl_name.ISD`	Data file
`tbl_name.ISM`	Index file

For more information on the properties of the various column types, see Section 7.2 [Column types], page 119.

- If neither `NULL` nor `NOT NULL` is specified, the column is treated as though `NULL` had been specified.
- An integer column may have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an `AUTO_INCREMENT` column, the column is set to `value+1`, where `value` is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`. See Section 20.4.29 [mysql_insert_id()], page 348.

  If you delete the row containing the maximum value for an `AUTO_INCREMENT` column, the value will be reused. If you delete all rows in the table, the sequence starts over.

  Note: There can be only one `AUTO_INCREMENT` column per table, and it must be indexed.

  To make **MySQL** compatible with some ODBC applications, you can find the last inserted row with the following query:

  ```
  SELECT * FROM tbl_name WHERE auto_col IS NULL
  ```

- `NULL` values are handled differently for `TIMESTAMP` columns than for other column types. You cannot store a literal `NULL` in a `TIMESTAMP` column; setting the column to `NULL` sets it to the current date and time. Because `TIMESTAMP` columns behave this way, the

`NULL` and `NOT NULL` attributes do not apply in the normal way and are ignored if you specify them.

On the other hand, to make it easier for **MySQL** clients to use `TIMESTAMP` columns, the server reports that such columns may be assigned `NULL` values (which is true), even though `TIMESTAMP` never actually will contain a `NULL` value. You can see this when you use `DESCRIBE tbl_name` to get a description of your table.

Note that setting a `TIMESTAMP` column to `0` is not the same as setting it to `NULL`, because `0` is a valid `TIMESTAMP` value.

- If no `DEFAULT` value is specified for a column, **MySQL** automatically assigns one.

  If the column may take `NULL` as a value, the default value is `NULL`.

  If the column is declared as `NOT NULL`, the default value depends on the column type:

  - For numeric types other than those declared with the `AUTO_INCREMENT` attribute, the default is `0`. For an `AUTO_INCREMENT` column, the default value is the next value in the sequence.
  - For date and time types other than `TIMESTAMP`, the default is the appropriate "zero" value for the type. For the first `TIMESTAMP` column in a table, the default value is the current date and time. See Section 7.2.3 [Date and time types], page 125.
  - For string types other than `ENUM`, the default is the empty string. For `ENUM`, the default is the first enumeration value.
- `KEY` is a synonym for `INDEX`.
- In **MySQL**, a `UNIQUE` key can have only distinct values. An error occurs if you try to add a new row with a key that matches an existing row.
- In **MySQL** a `PRIMARY KEY` is the same thing as a unique `KEY` that is named `PRIMARY`. A table can have only one `PRIMARY KEY`. If you don't have a `PRIMARY KEY` and some applications ask for the `PRIMARY KEY` in your tables, **MySQL** will return the first `UNIQUE` key as the `PRIMARY KEY`.
- A `PRIMARY KEY` can be a multiple-column index. However, you cannot create a multiple-column index using the `PRIMARY KEY` key attibute in a column specification. Doing so will mark only that single column as primary. You must use the `PRIMARY KEY(index_col_name, ...)` syntax.
- If you don't assign a name to an index, the index will be assigned the same name as the first `index_col_name`, with an optional suffix (`_2`, `_3`, `...`) to make it unique. You can see index names for a table using `SHOW INDEX FROM tbl_name`. See Section 7.20 [SHOW], page 190.
- Only the `MyISAM` table type supports indexes on columns that can have `NULL` values. In other cases you must declare such columns `NOT NULL` or an error results.
- With `col_name(length)` syntax, you can specify an index which uses only a part of a `CHAR` or `VARCHAR` column. This can make the index file much smaller. See Section 7.2.6 [Indexes], page 135.
- Only the `MyISAM` table type supports indexing on `BLOB` and `TEXT` columns. When putting an index on a `BLOB` or `TEXT` column you MUST always specify the length of the index:

```
CREATE TABLE test (blob_col BLOB, index(blob_col(10)));
```

- When you use `ORDER BY` or `GROUP BY` with a `TEXT` or `BLOB` column, only the first `max_sort_length` bytes are used. See Section 7.2.4.2 [BLOB], page 131.
- The `FOREIGN KEY`, `CHECK` and `REFERENCES` clauses don't actually do anything. The syntax for them is provided only for compatibility, to make it easier to port code from other SQL servers and to run applications that create tables with references. See Section 5.3 [Missing functions], page 88.
- Each `NULL` column takes one bit extra, rounded up to the nearest byte.
- The maximum record length in bytes can be calculated as follows:

```
row length = 1
             + (sum of column lengths)
             + (number of NULL columns + 7)/8
             + (number of variable-length columns)
```

- The `table_options` and `SELECT` options is only implemented in **MySQL** 3.23 and above.

  The different table types are:

ISAM	The original table handler
MyISAM	The new binary portable table handler
HEAP	The data for this table is only stored in memory

  See Section 10.18 [Table types], page 262.

  The other table options are used to optimize the behavior of the table. In most cases, you don't have to specify any of them. The options work for all table types, if not otherwise indicated.

`AUTO_INCREMENT`	The next auto_increment value you want to set for your table (MyISAM)
`AVG_ROW_LENGTH`	An approximation of the average row length for your table
`CHECKSUM`	Set this to 1 if you want **MySQL** to maintain a checksum for all rows (makes the table a little slower to update but makes it easier to find corrupted tables) (MyISAM)
`COMMENT`	A 60 character comment for your table
`MAX_ROWS`	Max number of rows you plan to store in the table
`MIN_ROWS`	Minimum number of rows you plan to store in the table
`PACK_KEYS`	Set this to 1 if you want to have smaller index. This usually makes updates slower and reads faster (MyISAM, ISAM).
`PASSWORD`	Encrypt the `.frm` file with a password. This option doesn't do anything in the standard **MySQL** version.
`DELAY_KEY_WRITE`	Set this to 1 if want to delay key table updates until the table is closed (MyISAM).

  When you use a `MyISAM` table, **MySQL** uses the product of `max_rows * avg_row_length` to decide how big the resulting table will be. If you don't specify any of the above options, the maximum size for a table will be 4G (or 2G if your operating systems only supports 2G tables).

- If you specify a `SELECT` after the `CREATE STATEMENT`, **MySQL** will create new fields for all elements in the `SELECT`. For example:

```
mysql> CREATE TABLE test (a int not null auto_increment,
           primary key (a), key(b))
           TYPE=HEAP SELECT b,c from test2;
```

  This will create a `HEAP` table with 3 columns. Note that the table will automatically be deleted if any errors occur while copying data into the table.

7.6.1 Silent column specification changes

In some cases, **MySQL** silently changes a column specification from that given in a `CREATE TABLE` statement. (This may also occur with `ALTER TABLE`.)

- `VARCHAR` columns with a length less than four are changed to `CHAR`.
- If any column in a table has a variable length, the entire row is variable-length as a result. Therefore, if a table contains any variable-length columns (`VARCHAR`, `TEXT` or `BLOB`), all `CHAR` columns longer than three characters are changed to `VARCHAR` columnss. This doesn't affect how you use the columns in any way; in **MySQL**, `VARCHAR` is just a different way to store characters. **MySQL** performs this conversion because it saves space and makes table operations faster. See Section 10.17 [Row format], page 260.
- `TIMESTAMP` display sizes must be even and in the range from 2 to 14. If you specify a display size of 0 or greater than 14, the size is coerced to 14. Odd-valued sizes in the range from 1 to 13 are coerced to the next higher even number.
- You cannot store a literal `NULL` in a `TIMESTAMP` column; setting it to `NULL` sets it to the current date and time. Because `TIMESTAMP` columns behave this way, the `NULL` and `NOT NULL` attributes do not apply in the normal way and are ignored if you specify them. `DESCRIBE tbl_name` always reports that a `TIMESTAMP` column may be assigned `NULL` values.
- **MySQL** maps certain column types used by other SQL database vendors to **MySQL** types. See Section 7.2.8 [Other-vendor column types], page 136.

If you want to see whether or not **MySQL** used a column type other than the one you specified, issue a `DESCRIBE tbl_name` statement after creating or altering your table.

Certain other column type changes may occur if you compress a table using `pack_isam`. See Section 10.17 [Row format], page 260.

7.7 ALTER TABLE syntax

```
ALTER [IGNORE] TABLE tbl_name alter_spec [, alter_spec ...]

alter_specification:
        ADD [COLUMN] create_definition [FIRST | AFTER column_name ]
  or    ADD INDEX [index_name] (index_col_name,...)
  or    ADD PRIMARY KEY (index_col_name,...)
  or    ADD UNIQUE [index_name] (index_col_name,...)
  or    ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}
```

```
or    CHANGE [COLUMN] old_col_name create_definition
or    MODIFY [COLUMN] create_definition
or    DROP [COLUMN] col_name
or    DROP PRIMARY KEY
or    DROP INDEX key_name
or    RENAME [AS] new_tbl_name
or    table_option
```

`ALTER TABLE` allows you to change the structure of an existing table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change the comment for the table and type of the table. See Section 7.6 [CREATE TABLE], page 167.

If you use `ALTER TABLE` to change a column specification but `DESCRIBE tbl_name` indicates that your column was not changed, it is possible that **MySQL** ignored your modification for one of the reasons described in Section 7.6.1 [Silent column changes], page 172. For example, if you try to change a `VARCHAR` column to `CHAR`, **MySQL** will still use `VARCHAR` if the table contains other variable-length columns.

`ALTER TABLE` works by making a temporary copy of the original table. The alteration is performed on the copy, then the original table is deleted and the new one is renamed. This is done in such a way that all updates are automatically redirected to the new table without any failed updates. While `ALTER TABLE` is executing, the original table is readable by other clients. Updates and writes to the table are stalled until the new table is ready.

- To use `ALTER TABLE`, you need **select**, **insert**, **delete**, **update**, **create** and **drop** privileges on the table.
- `IGNORE` is a **MySQL** extension to ANSI SQL92. It controls how `ALTER TABLE` works if there are duplicates on unique keys in the new table. If `IGNORE` isn't specified, the copy is aborted and rolled back. If `IGNORE` is specified, then for rows with duplicates on a unique key, only the first row is used; the others are deleted.
- You can issue multiple `ADD`, `ALTER`, `DROP` and `CHANGE` clauses in a single `ALTER TABLE` statement. This is a **MySQL** extension to ANSI SQL92, which allows only one of each clause per `ALTER TABLE` statement.
- `CHANGE col_name`, `DROP col_name` and `DROP INDEX` are **MySQL** extensions to ANSI SQL92.
- `MODIFY` is an Oracle extension to `ALTER TABLE`.
- The optional word `COLUMN` is a pure noise word and can be omitted.
- If you use `ALTER TABLE tbl_name RENAME AS new_name` without any other options, **MySQL** simply renames the files that correspond to the table `tbl_name`. There is no need to create the temporary table.
- `create_definition` clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. Note that this syntax includes the column name, not just the column type. See Section 7.6 [CREATE TABLE], page 167.
- You can rename a column using a `CHANGE old_col_name create_definition` clause. To do so, specify the old and new column names and the type that the column currently has. For example, to rename an `INTEGER` column from `a` to `b`, you can do this:

```
mysql> ALTER TABLE t1 CHANGE a b INTEGER;
```

If you want to change a column's type but not the name, `CHANGE` syntax still requires two column names even if they are the same. For example:

```
mysql> ALTER TABLE t1 CHANGE b b BIGINT NOT NULL;
```

However, as of **MySQL** 3.22.16a, you can also use `MODIFY` to change a column's type without renaming it:

```
mysql> ALTER TABLE t1 MODIFY b BIGINT NOT NULL;
```

- If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on part of the column (for instance, if you have an index on the first 10 characters of a `VARCHAR` column), you cannot make the column shorter than the number of characters that are indexed.
- When you change a column type using `CHANGE` or `MODIFY`, **MySQL** tries to convert data to the new type as well as possible.
- In **MySQL** 3.22 or later, you can use `FIRST` or `ADD ... AFTER col_name` to add a column at a specific position within a table row. The default is to add the column last.
- `ALTER COLUMN` specifies a new default value for a column or removes the old default value. If the old default is removed and the column can be `NULL`, the new default is `NULL`. If the column cannot be `NULL`, **MySQL** assigns a default value. Default value assignment is described in Section 7.6 [CREATE TABLE], page 167.
- `DROP INDEX` removes an index. This is a **MySQL** extension to ANSI SQL92.
- If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well.
- `DROP PRIMARY KEY` drops the primary index. If no such index exists, it drops the first `UNIQUE` index in the table. (**MySQL** marks the first `UNIQUE` key as the `PRIMARY KEY` if no `PRIMARY KEY` was specified explicitly.)
- With the C API function `mysql_info()`, you can find out how many records were copied, and (when `IGNORE` is used) how many records were deleted due to duplication of unique key values.
- The `FOREIGN KEY`, `CHECK` and `REFERENCES` clauses don't actually do anything. The syntax for them is provided only for compatibility, to make it easier to port code from other SQL servers and to run applications that create tables with references. See Section 5.3 [Missing functions], page 88.

Here is an example that shows some of the uses of `ALTER TABLE`. We begin with a table `t1` that is created as shown below:

```
mysql> CREATE TABLE t1 (a INTEGER,b CHAR(10));
```

To rename the table from `t1` to `t2`:

```
mysql> ALTER TABLE t1 RENAME t2;
```

To change column `a` from `INTEGER` to `TINYINT NOT NULL` (leaving the name the same), and to change column `b` from `CHAR(10)` to `CHAR(20)` as well as renaming it from `b` to `c`:

```
mysql> ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new `TIMESTAMP` column named `d`:

```
mysql> ALTER TABLE t2 ADD d TIMESTAMP;
```

To add an index on column `d`, and make column `a` the primary key:

```
mysql> ALTER TABLE t2 ADD INDEX (d), ADD PRIMARY KEY (a);
```

To remove column `c`:

```
mysql> ALTER TABLE t2 DROP COLUMN c;
```

To add a new `AUTO_INCREMENT` integer column named `c`:

```
mysql> ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
           ADD INDEX (c);
```

Note that we indexed `c`, because `AUTO_INCREMENT` columns must be indexed, and also that we declare `c` as `NOT NULL`, because indexed columns cannot be `NULL`.

When you add an `AUTO_INCREMENT` column, column values are filled in with sequence numbers for you automatically.

7.8 OPTIMIZE TABLE syntax

```
OPTIMIZE TABLE tbl_name
```

`OPTIMZE TABLE` should be used if you have deleted a large part of a table or if you have made many changes to a table with variable-length rows (tables that have `VARCHAR`, `BLOB` or `TEXT` columns). Deleted records are maintained in a linked list and subsequent `INSERT` operations reuse old record positions. You can use `OPTIMIZE TABLE` to reclaim the unused space.

`OPTIMIZE TABLE` works by making a temporary copy of the original table. The old table is copied to the new table (without the unused rows), then the original table is deleted and the new one is renamed. This is done in such a way that all updates are automatically redirected to the new table without any failed updates. While `OPTIMIZE TABLE` is executing, the original table is readable by other clients. Updates and writes to the table are stalled until the new table is ready.

7.9 DROP TABLE syntax

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

`DROP TABLE` removes one or more tables. All table data and the table definition are *removed*, so **be careful** with this command!

In **MySQL** 3.22 or later, you can use the keywords `IF EXISTS` to prevent an error from occurring for tables that don't exist.

7.10 DELETE syntax

```
DELETE [LOW_PRIORITY] FROM tbl_name
    [WHERE where_definition] [LIMIT rows]
```

`DELETE` deletes rows from `tbl_name` that satisfy the condition given by `where_definition`, and returns the number of records deleted.

If you issue a `DELETE` with no `WHERE` clause, all rows are deleted. **MySQL** does this by recreating the table as an empty table, which is much faster than deleting each row. In this case, `DELETE` returns zero as the number of affected records. (**MySQL** can't return the number of rows that were actually deleted, since the recreate is done without opening the data files. As long as the table definition file '`tbl_name.frm`' is valid, the table can be recreated this way, even if the data or index files have become corrupted.).

If you really want to know how many records are deleted when you are deleting all rows, and are willing to suffer a speed penalty, you can use a `DELETE` statement of this form:

```
mysql> DELETE FROM tbl_name WHERE 1>0;
```

Note that this is MUCH slower than `DELETE FROM tbl_name` with no `WHERE` clause, because it deletes rows one at a time.

If you specify the keyword `LOW_PRIORITY`, execution of the `DELETE` is delayed until no other clients are reading from the table.

Deleted records are maintained in a linked list and subsequent `INSERT` operations reuse old record positions. To reclaim unused space and reduce file sizes, use the `OPTIMIZE TABLE` statement or the `isamchk` utility to reorganize tables. `OPTIMIZE TABLE` is easier, but `isamchk` is faster. See Section 7.8 [OPTIMIZE TABLE], page 175, and Section 13.4.3 [Optimization], page 292.

The **MySQL**-specific `LIMIT rows` option to `DELETE` tells the server the maximum number of rows to be deleted before control is returned to the client. This can be used to ensure that a specific `DELETE` command doesn't take too much time. You can simply repeat the `DELETE` command until the number of affected rows is less than the `LIMIT` value.

7.11 SELECT syntax

```
SELECT [STRAIGHT_JOIN] [SQL_SMALL_RESULT] [DISTINCT | DISTINCTROW | ALL]
    select_expression,...
    [INTO OUTFILE 'file_name' export_options]
    [FROM table_references
        [WHERE where_definition]
        [GROUP BY col_name,...]
        [HAVING where_definition]
        [ORDER BY {unsigned_integer | col_name | formula} [ASC | DESC] ,...]
        [LIMIT [offset,] rows]
        [PROCEDURE procedure_name] ]
```

`SELECT` is used to retrieve rows selected from one or more tables. `select_expression` indicates the columns you want to retrieve. `SELECT` may also be used to retrieve rows computed without reference to any table. For example:

```
mysql> SELECT 1 + 1;
         -> 2
```

All keywords used must be given in exactly the order shown above. For example, a `HAVING` clause must come after any `GROUP BY` clause and before any `ORDER BY` clause.

- A `SELECT` expression may be given an alias using `AS`. The alias is used as the expression's column name and can be used with `ORDER BY` or `HAVING` clauses. For example:

```
mysql> select concat(last_name,', ',first_name) AS full_name
    from mytable ORDER BY full_name;
```

- The `FROM table_references` clause indicates the tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see Section 7.12 [JOIN], page 178.
- You can refer to a column as `col_name`, `tbl_name.col_name` or `db_name.tbl_name.col_name`. You need not specify a `tbl_name` or `db_name.tbl_name` prefix for a column reference in a `SELECT` statement unless the reference would be ambiguous. See Section 7.1.5 [Legal names], page 118, for examples of ambiguity that require the more explicit column reference forms.
- A table reference may be aliased using `tbl_name [AS] alias_name`.

```
mysql> select t1.name, t2.salary from employee AS t1, info AS t2
           where t1.name = t2.name;
mysql> select t1.name, t2.salary from employee t1, info t2
           where t1.name = t2.name;
```

- Columns selected for output may be referred to in `ORDER BY` and `GROUP BY` clauses using column names, column aliases or column positions. Column positions begin with 1.

```
mysql> select college, region, seed from tournament
           ORDER BY region, seed;
mysql> select college, region AS r, seed AS s from tournament
           ORDER BY r, s;
mysql> select college, region, seed from tournament
           ORDER BY 2, 3;
```

  To sort in reverse order, add the `DESC` (descending) keyword to the name of the column in the `ORDER BY` clause that you are sorting by. The default is ascending order; this may be specified explicitly using the `ASC` keyword.
- The `HAVING` clause can refer to any column or alias named in the `select_expression`. It is applied last, just before items are sent to the client, with no optimization. Don't use `HAVING` for items that should be in the `WHERE` clause. For example, do not write this:

```
mysql> select col_name from tbl_name HAVING col_name > 0;
```

  Write this instead:

```
mysql> select col_name from tbl_name WHERE col_name > 0;
```

  In **MySQL** 3.22.5 or later, you can also write queries like this:

```
mysql> select user,max(salary) from users
           group by user HAVING max(salary)>10;
```

  In older **MySQL** versions, you can write this instead:

```
mysql> select user,max(salary) AS sum from users
           group by user HAVING sum>10;
```

- `STRAIGHT_JOIN` forces the optimizer to join the tables in the order in which they are listed in the `FROM` clause. You can use this to speed up a query if the optimizer joins the tables in non-optimal order. See Section 7.21 [EXPLAIN], page 194.
- `SQL_SMALL_RESULT` can be used with `GROUP BY` or `DISTINCT` to tell the optimizer that the result set will be small. In this case, **MySQL** will use fast temporary tables to store

the resulting table instead of using sorting. `SQL_SMALL_RESULT` is a **MySQL** extension to ANSI SQL92.

- The `LIMIT` clause can be used to constrain the number of rows returned by the `SELECT` statement. `LIMIT` takes one or two numeric arguments.

  If two arguments are given, the first specifies the offset of the first row to return, the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1).

  ```
  mysql> select * from table LIMIT 5,10;  # Retrieve rows 6-15
  ```

  If one argument is given, it indicates the maximum number of rows to return.

  ```
  mysql> select * from table LIMIT 5;     # Retrieve first 5 rows
  ```

  In other words, `LIMIT n` is equivalent to `LIMIT 0,n`.

- The `SELECT ... INTO OUTFILE 'file_name'` form of `SELECT` writes the selected rows to a file. The file is created on the server host, and cannot already exist (among other things, this prevents database tables and files such as '`/etc/passwd`' from being destroyed). You must have the **file** privilege on the server host to use this form of `SELECT`.

  `SELECT ... INTO OUTFILE` is the complement of `LOAD DATA INFILE`; the syntax for the `export_options` part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA INFILE` statement. See Section 7.15 [LOAD DATA], page 183.

  In the resulting text file, only the following characters are escaped by the `ESCAPED BY` character:

  - The `ESCAPED BY` character
  - The first character in `FIELDS TERMINATED BY`
  - The first character in `LINES TERMINATED BY`

  Additionally, `ASCII 0` is converted to `ESCAPED BY` followed by 0 (`ASCII 48`).

  The reason for the above is that you MUST escape any `FIELDS TERMINATED BY`, `ESCAPED BY` or `LINES TERMINATED BY` characters to reliably be able to read the file back. `ASCII 0` is escaped to make it easier to view with some pagers.

  As the resulting file doesn't have to conform to the SQL syntax, nothing else need be escaped.

7.12 `JOIN` syntax

MySQL supports the following `JOIN` syntaxes for use in `SELECT` statements:

```
table_reference, table_reference
table_reference [CROSS] JOIN table_reference
table_reference INNER JOIN table_reference
table_reference STRAIGHT_JOIN table_reference
table_reference LEFT [OUTER] JOIN table_reference ON conditional_expr
table_reference LEFT [OUTER] JOIN table_reference USING (column_list)
table_reference NATURAL LEFT [OUTER] JOIN table_reference
```

```
{ oj table_reference LEFT OUTER JOIN table_reference ON conditional_expr }
```

The last `LEFT OUTER JOIN` syntax shown above exists only for compatibility with ODBC.

- A table reference may be aliased using `tbl_name AS alias_name` or `tbl_name alias_name`.

  ```
  mysql> select t1.name, t2.salary from employee AS t1, info AS t2
             where t1.name = t2.name;
  ```

- `INNER JOIN` and `,` (comma) are semantically equivalent. Both do a full join between the tables used. Normally, you specify how the tables should be linked in the `WHERE` condition.
- The `ON` conditional is any conditional of the form that may be used in a `WHERE` clause.
- If there is no matching record for the right table in a `LEFT JOIN`, a row with all columns set to `NULL` is used for the right table. You can use this fact to find records in a table that have no counterpart in another table:

  ```
  mysql> select table1.* from table1
             LEFT JOIN table2 ON table1.id=table2.id
             where table2.id is NULL;
  ```

  This example finds all rows in `table1` with an `id` value that is not present in `table2` (i.e., all rows in `table1` with no corresponding row in `table2`). This assumes that `table2.id` is declared `NOT NULL`, of course.
- The `USING column_list` clause names a list of columns that must exist in both tables. A `USING` clause such as:

  ```
  A LEFT JOIN B USING (C1,C2,C3,...)
  ```

  is defined to be semantically identical to an `ON` expression like this:

  ```
  A.C1=B.C1 AND A.C2=B.C2 AND A.C3=B.C3,...
  ```

- The `NATURAL LEFT JOIN` of two tables is defined to be semantically equivalent to a `LEFT JOIN` with a `USING` clause that names all columns that exist in both tables.
- `STRAIGHT_JOIN` is identical to `JOIN`, except that the left table is always read before the right table. This can be used for those (few) cases where the join optimizer puts the tables in the wrong order.

Some examples:

```
mysql> select * from table1,table2 where table1.id=table2.id;
mysql> select * from table1 LEFT JOIN table2 ON table1.id=table2.id;
mysql> select * from table1 LEFT JOIN table2 USING (id);
mysql> select * from table1 LEFT JOIN table2 ON table1.id=table2.id
          LEFT JOIN table3 ON table2.id=table3.id;
```

See Section 10.6 [LEFT JOIN optimization], page 253.

7.13 INSERT syntax

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
    [INTO] tbl_name [(col_name,...)]
    VALUES (expression,...),(...),...
or  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
```

```
        [INTO] tbl_name [(col_name,...)]
        SELECT ...
or  INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
        [INTO] tbl_name
        SET col_name=expression, col_name=expression, ...
```

`INSERT` inserts new rows into an existing table. The `INSERT ... VALUES` form of the statement inserts rows based on explicitly-specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. The `INSERT ... VALUES` form with multiple value lists is supported in **MySQL** 3.22.5 or later. The `col_name=expression` syntax is supported in **MySQL** 3.22.10 or later.

`tbl_name` is the table into which rows should be inserted. The column name list or the `SET` clause indicates which columns the statement specifies values for.

- If you specify no column list for `INSERT ... VALUES` or `INSERT ... SELECT`, values for all columns must be provided in the `VALUES()` list or by the `SELECT`. If you don't know the order of the columns in the table, use `DESCRIBE tbl_name` to find out.
- Any column not explicitly given a value is set to its default value. For example, if you specify a column list that doesn't name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in Section 7.6 [CREATE TABLE], page 167.
- An `expression` may refer to any column that was set earlier in a value list. For example, you can say this:

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
  ```

  But not this:

  ```
  mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
  ```

- If you specify the keyword `LOW_PRIORITY`, execution of the `INSERT` is delayed until no other clients are reading from the table.
- If you specify the keyword `IGNORE` in an `INSERT` with many value rows, any rows which duplicate an existing `PRIMARY` or `UNIQUE` key in the table are ignored and are not inserted. If you do not specify `IGNORE`, the insert is aborted if there is any row that duplicates an existing key value. You can check with the `mysql_info()` how many rows were inserted into the table.
- If **MySQL** was configured using the `DONT_USE_DEFAULT_FIELDS` option, `INSERT` statements generate an error unless you explicitly specify values for all columns that require a non-`NULL` value. See Section 4.7.3 [configure options], page 42.
- The following conditions hold for a `INSERT INTO ... SELECT` statement:
  - The query cannot contain an `ORDER BY` clause.
  - The target table of the `INSERT` statement cannot appear in the `FROM` clause of the `SELECT` part of the query, because it's forbidden in ANSI SQL to `SELECT` from the same table into which you are `INSERT`ing. (The problem is that the `SELECT` possibly would find records that were inserted earlier during the same run. When using sub-select clauses, the situation could easily be very confusing!)
  - `AUTO_INCREMENT` columns work as usual.

If you use `INSERT ... SELECT` or a `INSERT ... VALUES` statement with multiple value lists, you can use the C API function `mysql_info()` to get information about the query. The format of the information string is shown below:

```
Records: 100 Duplicates: 0 Warnings: 0
```

`Duplicates` indicates the number of rows that couldn't be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. The column is set to its default value.
- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the appropriate endpoint of the range.
- Setting a numeric column to a value such as `'10.34 a'`. The trailing garbage is stripped and the remaining numeric part is inserted. If the value doesn't make sense as a number at all, the column is set to `0`.
- Inserting a string into a `CHAR`, `VARCHAR`, `TEXT` or `BLOB` column that exceeds the column's maximum length. The value is truncated to the column's maximum length.
- Inserting a value into a date or time column that is illegal for the column type. The column is set to the appropriate "zero" value for the type.

The `DELAYED` option for the `INSERT` statement is a **MySQL**-specific option that is very useful if you have clients that can't wait for the `INSERT` to complete. This is common when you use **MySQL** for logging and you also periodically run `SELECT` statements that take a long time to complete. `DELAYED` was introduced in **MySQL** 3.22.15. It is a **MySQL** extension to ANSI SQL92.

Another major benefit of using `INSERT DELAYED` is that inserts from many clients are bundled together and written in one block. This is much faster than doing many separate inserts.

Note that currently the queued rows are only stored in memory until they are inserted into the table. This means that if you kill `mysqld` the hard way (`kill -9`) or if `mysqld` dies unexpectedly, any queued rows that weren't written to disk are lost!

The following happens when you use the `DELAYED` option to `INSERT` or `REPLACE`. In this description, the "thread" is the thread that received an `INSERT DELAYED` command and "handler" is the thread that handles all `INSERT DELAYED` statements for a particular table.

- When a thread executes a `DELAYED` statement for a table, a handler thread is created to handle all `DELAYED` statements for the table, if no such handler already exists.
- The thread checks whether or not the handler has acquired a `DELAYED` lock already; if not, it tells the handler thread to do so. The `DELAYED` lock can be obtained even if other threads have a `READ` or `WRITE` lock on the table. However, the handler will wait for all `ALTER TABLE` locks or `FLUSH TABLES` to ensure that the table structure is up to date.
- The thread executes the `INSERT` statement but instead of writing the row to the table it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported the client program.

- The client can't report the number of duplicates or the `AUTO_INCREMENT` value for the resulting row; it can't obtain them from the server, because the `INSERT` returns before the insert operation has been completed. If you use the C API, the `mysql_info()` function doesn't return anything meaningful, for the same reason.
- The update log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the update log is updated when the first row is inserted.
- After every `delayed_insert_limit` rows are written, the handler checks whether or not any `SELECT` statements are still pending. If so, it allows these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new `INSERT DELAYED` commands are received within `delayed_insert_timeout` seconds, the handler terminates.
- If more than `delayed_queue_size` rows are pending already in a specific handler queue, the thread waits until there is room in the queue. This is useful to ensure that the `mysqld` server doesn't use all memory for the delayed memory queue.
- The handler thread will show up in the **MySQL** process list with `delayed_insert` in the `Command` column. It will be killed if you execute a `FLUSH TABLES` command or kill it with `KILL thread_id`. However, it will first store all queued rows into the table before exiting. During this time it will not accept any new `INSERT` commands from another thread. If you execute an `INSERT DELAYED` command after this, a new handler thread will be created.
- Note that the above means that `INSERT DELAYED` commands have higher priority than normal `INSERT` commands if there is an `INSERT DELAYED` handler already running! Other update commands will have to wait until the `INSERT DELAY` queue is empty, someone kills the handler thread (with `KILL thread_id`) or someone executes `FLUSH TABLES`.
- The following status variables provide information about `INSERT DELAYED` commands:

`Delayed_insert_threads`	Number of handler threads
`Delayed_writes`	Number of rows written with `INSERT DELAYED`
`Not_flushed_delayed_rows`	Number of rows waiting to be written

  You can view these variables by issuing a `SHOW STATUS` statement or by executing a `mysqladmin extended-status` command.

7.14 REPLACE syntax

```
    REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name [(col_name,...)]
        VALUES (expression,...)
or  REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name [(col_name,...)]
        SELECT ...
or  REPLACE [LOW_PRIORITY | DELAYED]
        [INTO] tbl_name
```

```
SET col_name=expression, col_name=expression,...
```

`REPLACE` works exactly like `INSERT`, except that if an old record in the table has the same value as a new record on a unique index, the old record is deleted before the new record is inserted. See Section 7.13 [INSERT], page 179.

7.15 LOAD DATA INFILE syntax

```
LOAD DATA [LOCAL] INFILE 'file_name.txt' [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [FIELDS
        [TERMINATED BY '\t']
        [OPTIONALLY] ENCLOSED BY '']
        [ESCAPED BY '\\' ]]
    [LINES TERMINATED BY '\n']
    [IGNORE number LINES]
    [(col_name,...)]
```

The `LOAD DATA INFILE` statement reads rows from a text file into a table at a very high speed. If the `LOCAL` keyword is specified, the file is read from the client host. If `LOCAL` is not specified, the file must be located on the server. (`LOCAL` is available in **MySQL** 3.22.6 or later.)

For security reasons, when reading text files located on the server, the files must either reside in the database directory or be readable by all. Also, to use `LOAD DATA INFILE` on server files, you must have the **file** privilege on the server host. See Section 6.5 [Privileges provided], page 96.

Using `LOCAL` will be a bit slower than letting the server access the files directly, since the contents of the file must travel from the client host to the server host. On the other hand, you do not need the **file** privilege to load local files.

You can also load data files by using the `mysqlimport` utility; it operates by sending a `LOAD DATA INFILE` command to the server. The `--local` option causes `mysqlimport` to read data files from the client host. You can specify the `--compress` option to get better performance over slow networks if the client and server support the compressed protocol.

When locating files on the server host, the server uses the following rules:

- If an absolute pathname is given, the server uses the pathname as is.
- If a relative pathname with one or more leading components is given, the server searches for the file relative to the server's data directory.
- If a filename with no leading components is given, the server looks for the file in the database directory of the current database.

Note that these rules mean a file given as '`./myfile.txt`' is read from the server's data directory, whereas a file given as '`myfile.txt`' is read from the database directory of the current database. Note also that for statements such as those below, the file is read from the database directory for `db1`, not `db2`:

```
mysql> USE db1;
mysql> LOAD DATA INFILE "./data.txt" INTO TABLE db2.my_table;
```

The REPLACE and IGNORE keywords control handling of input records that duplicate existing records on unique key values. If you specify REPLACE, new rows replace existing rows that have the same unique key value. If you specify IGNORE, input rows that duplicate an existing row on a unique key value are skipped. If you don't specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

If you load data from a local file using the LOCAL keyword, the server has no way to stop transmission of the file in the middle of the operation, so the default bahavior is the same as if IGNORE is specified.

LOAD DATA INFILE is the complement of SELECT ... INTO OUTFILE. See Section 7.11 [SELECT], page 176. To write data from a database to a file, use SELECT ... INTO OUTFILE. To read the file back into the database, use LOAD DATA INFILE. The syntax of the FIELDS and LINES clauses is the same for both commands. Both clauses are optional, but FIELDS must precede LINES if both are specified.

If you specify a FIELDS clause, each of its subclauses (TERMINATED BY, [OPTIONALLY] ENCLOSED BY and ESCAPED BY) is also optional, except that you must specify at least one of them.

If you don't specify a FIELDS clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
```

If you don't specify a LINES clause, the default is the same as if you had written this:

```
LINES TERMINATED BY '\n'
```

In other words, the defaults cause LOAD DATA INFILE to act as follows when reading input:

- Look for line boundaries at newlines
- Break lines into fields at tabs
- Do not expect fields to be enclosed within any quoting characters
- Interpret occurrences of tab, newline or '\' preceded by '\' as literal characters that are part of field values

Conversely, the defaults cause SELECT ... INTO OUTFILE to act as follows when writing output:

- Write tabs between fields
- Do not enclose fields within any quoting characters
- Use '\' to escape instances of tab, newline or '\' that occur within field values
- Write newlines at the ends of lines

Note that to write FIELDS ESCAPED BY '\\', you must specify two backslashes for the value to be read as a single backslash.

The IGNORE number LINES option can be used to ignore a header of column names at the start of the file:

```
mysql> LOAD DATA INFILE "/tmp/file_name" into table test IGNORE 1 LINES;
```

When you use SELECT ... INTO OUTFILE in tandem with LOAD DATA INFILE to write data from a database into a file and then read the file back into the database later, the field and line handling options for both commands must match. Otherwise, LOAD DATA INFILE will not interpret the contents of the file properly. Suppose you use SELECT ... INTO OUTFILE to write a file with fields delimited by commas:

```
mysql> SELECT * FROM table1 INTO OUTFILE 'data.txt'
           FIELDS TERMINATED BY ','
           FROM ...
```

To read the comma-delimited file back in, the correct statement would be:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
           FIELDS TERMINATED BY ',';
```

If instead you tried to read in the file with the statement shown below, it wouldn't work because it instructs `LOAD DATA INFILE` to look for tabs between fields:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE table2
           FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

`LOAD DATA INFILE` can be used to read files obtained from external sources, too. For example, a file in dBASE format will have fields separated by commas and enclosed in double quotes. If lines in the file are terminated by newlines, the command shown below illustrates the field and line handling options you would use to load the file:

```
mysql> LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
           FIELDS TERMINATED BY ',' ENCLOSED BY '"'
           LINES TERMINATED BY '\n';
```

Any of the field or line handling options may specify an empty string (`''`). If not empty, the `FIELDS [OPTIONALLY] ENCLOSED BY` and `FIELDS ESCAPED BY` values must be a single character. The `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values may be more than one character. For example, to write lines that are terminated by carriage return-linefeed pairs, or to read a file containing such lines, specify a `LINES TERMINATED BY '\r\n'` clause.

`FIELDS [OPTIONALLY] ENCLOSED BY` controls quoting of fields. For output (`SELECT ... INTO OUTFILE`), if you omit the word `OPTIONALLY`, all fields are enclosed by the `ENCLOSED BY` character. An example of such output (using a comma as the field delimiter) is shown below:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify `OPTIONALLY`, the `ENCLOSED BY` character is used only to enclose `CHAR` and `VARCHAR` fields:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Note that occurrences of the `ENCLOSED BY` character within a field value are escaped by prefixing them with the `ESCAPED BY` character. Also note that if you specify an empty `ESCAPED BY` value, it is possible to generate output that cannot be read properly by `LOAD DATA INFILE`. For example, the output just shown above would appear as shown below if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the `ENCLOSED BY` character, if present, is stripped from the ends of field values. (This is true whether or not `OPTIONALLY` is specified; `OPTIONALLY` has no effect on input interpretation.) Occurrences of the `ENCLOSED BY` character preceded by the `ESCAPED BY` character are interpreted as part of the current field value. In addition, duplicated `ENCLOSED BY` characters occurring within fields are interpreted as single `ENCLOSED BY` characters if the field itself starts with that character. For example, if `ENCLOSED BY '"'` is specified, quotes are handled as shown below:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

`FIELDS ESCAPED BY` controls how to write or read special characters. If the `FIELDS ESCAPED BY` character is not empty, it is used to prefix the following characters on output:

- The `FIELDS ESCAPED BY` character
- The `FIELDS [OPTIONALLY] ENCLOSED BY` character
- The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
- ASCII `0` (what is actually written following the escape character is ASCII `'0'`, not a zero-valued byte)

If the `FIELDS ESCAPED BY` character is empty, no characters are escaped. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

For input, if the `FIELDS ESCAPED BY` character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. The exceptions are an escaped '`0`' or '`N`' (e.g., `\0` or `\N` if the escape character is '`\`'). These sequences are interpreted as ASCII `0` (a zero-valued byte) and `NULL`. See below for the rules on `NULL` handling.

For more information about '`\`'-escape syntax, see Section 7.1 [Literals], page 116.

In certain cases, field and line handling options interact:

- If `LINES TERMINATED BY` is an empty string and `FIELDS TERMINATED BY` is non-empty, lines are also terminated with `FIELDS TERMINATED BY`.
- If the `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` values are both empty (`''`), a fixed-row (non-delimited) format is used. With fixed-row format, no delimiters are used between fields. Instead, column values are written and read using the "display" widths of the columns. For example, if a column is declared as `INT(7)`, values for the column are written using 7-character fields. On input, values for the column are obtained by reading 7 characters. Fixed-row format also affects handling of `NULL` values; see below.

Handling of `NULL` values varies, depending on the `FIELDS` and `LINES` options you use:

- For the default `FIELDS` and `LINES` values, `NULL` is written as `\N` for output and `\N` is read as `NULL` for input (assuming the `ESCAPED BY` character is '`\`').

- If `FIELDS ENCLOSED BY` is not empty, a field containing the literal word `NULL` as its value is read as a `NULL` value (this differs from the word `NULL` enclosed within `FIELDS ENCLOSED BY` characters, which is read as the string `'NULL'`).
- If `FIELDS ESCAPED BY` is empty, `NULL` is written as the word `NULL`.
- With fixed-row format (which happens when `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` are both empty), `NULL` is written as an empty string. Note that this causes both `NULL` values and empty strings in the table to be indistinguishable when written to the file since they are both written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

Some cases are not supported by `LOAD DATA INFILE`:

- Fixed-size rows (`FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` both empty) and `BLOB` or `TEXT` columns.
- If you specify one separator that is the same as or a prefix of another, `LOAD DATA INFILE` won't be able to interpret the input properly. For example, the following `FIELDS` clause would cause problems:

```
FIELDS TERMINATED BY '"' ENCLOSED BY '"'
```

- If `FIELDS ESCAPED BY` is empty, a field value that contains an occurrence of `FIELDS ENCLOSED BY` or `LINES TERMINATED BY` followed by the `FIELDS TERMINATED BY` value will cause `LOAD DATA INFILE` to stop reading a field or line too early. This happens because `LOAD DATA INFILE` cannot properly determine where the field or line value ends.

The following example loads all columns of the `persondata` table:

```
mysql> LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

No field list is specified, so `LOAD DATA INFILE` expects input rows to contain a field for each table column. The default `FIELDS` and `LINES` values are used.

If you wish to load only some of a table's columns, specify a field list:

```
mysql> LOAD DATA INFILE 'persondata.txt'
           INTO TABLE persondata (col1,col2,...);
```

You must also specify a field list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, **MySQL** cannot tell how to match up input fields with table columns.

If a row has too few fields, the columns for which no input field is present are set to default values. Default value assignment is described in Section 7.6 [CREATE TABLE], page 167.

An empty field value is interpreted differently than if the field value is missing:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to `0`.
- For date and time types, the column is set to the appropriate "zero" value for the type. See Section 7.2.3 [Date and time types], page 125.

`TIMESTAMP` columns are only set to the current date and time if there is a `NULL` value for the column, or (for the first `TIMESTAMP` column only) if the `TIMESTAMP` column is left out from the field list when a field list is specified.

If an input row has too many fields, the extra fields are ignored and the number of warnings is incremented.

`LOAD DATA INFILE` regards all input as strings, so you can't use numeric values for `ENUM` or `SET` columns the way you can with `INSERT` statements. All `ENUM` and `SET` values must be specified as strings!

If you are using the C API, you can get information about the query by calling the API function `mysql_info()` when the `LOAD DATA INFILE` query finishes. The format of the information string is shown below:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

Warnings occur under the same circumstances as when values are inserted via the `INSERT` statement (see Section 7.13 [INSERT], page 179), except that `LOAD DATA INFILE` also generates warnings when there are too few or too many fields in the input row. The warnings are not stored anywhere; the number of warnings can only be used as an indication if everything went well. If you get warnings and want to know exactly why you got them, one way to do this is to use `SELECT ... INTO OUTFILE` into another file and compare this to your original input file.

For more information about the efficiency of `INSERT` versus `LOAD DATA INFILE` and speeding up `LOAD DATA INFILE`, see Section 10.12 [Table efficiency], page 255.

7.16 `UPDATE` syntax

```
UPDATE [LOW_PRIORITY] tbl_name SET col_name1=expr1,col_name2=expr2,...
    [WHERE where_definition] [LIMIT #]
```

`UPDATE` updates columns in existing table rows with new values. The `SET` clause indicates which columns to modify and the values they should be given. The `WHERE` clause, if given, specifies which rows should be updated. Otherwise all rows are updated.

If you specify the keyword `LOW_PRIORITY`, execution of the `UPDATE` is delayed until no other clients are reading from the table.

If you access a column from `tbl_name` in an expression, `UPDATE` uses the current value of the column. For example, the following statement sets the `age` column to one more than its current value:

```
mysql> UPDATE persondata SET age=age+1;
```

`UPDATE` assignments are evaluated from left to right. For example, the following statement doubles the `age` column, then increments it:

```
mysql> UPDATE persondata SET age=age*2, age=age+1;
```

If you set a column to the value it currently has, **MySQL** notices this and doesn't update it.

`UPDATE` returns the number of rows that were actually changed. In **MySQL** 3.22 or later, the C API function `mysql_info()` returns the number of rows that were matched and updated and the number of warnings that occurred during the `UPDATE`.

In **MySQL** 3.23 you can use `LIMIT #` to ensure that only a given number of rows are changed.

7.17 USE syntax

```
USE db_name
```

The `USE db_name` statement tells **MySQL** to use the `db_name` database as the default database for subsequent queries. The database remains current until the end of the session, or until another `USE` statement is issued:

```
mysql> USE db1;
mysql> SELECT count(*) FROM mytable;      # selects from db1.mytable
mysql> USE db2;
mysql> SELECT count(*) FROM mytable;      # selects from db2.mytable
```

Making a particular database current by means of the `USE` statement does not preclude you from accessing tables in other databases. The example below accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
mysql> USE db1;
mysql> SELECT author_name,editor_name FROM author,db2.editor
           WHERE author.editor_id = db2.editor.editor_id;
```

The `USE` statement is provided for Sybase compatibility.

7.18 FLUSH syntax (clearing caches)

```
FLUSH flush_option [,flush_option]
```

You should use the `FLUSH` command if you want to clear some of the internal caches **MySQL** uses. To execute `FLUSH`, you must have the **reload** privilege.

`flush_option` can be any of the following:

`HOSTS` Empties the host cache tables. You should flush the host tables if some of your hosts change IP number or if you get the error message `Host ... is blocked`. When more than `max_connect_errors` errors occur in a row for a given host while connection to the **MySQL** server, **MySQL** assumes something is wrong and blocks the host from further connection requests. Flushing the host tables allows the host to attempt to connect again. See Section 18.2.3 [Blocked host], page 310.) You can start `mysqld` with `-O max_connection_errors=999999999` to avoid this error message.

`LOGS` Closes and reopens the standard and update log files. If you have specified the update log file without an extension, the extension number of the new update log file will be incremented by one relative to the previous file.

`PRIVILEGES` Reloads the privileges from the grant tables in the `mysql` database.

`TABLES` Closes all open tables.

`STATUS` Resets most status variables to zero.

You can also access each of the commands shown above with the `mysqladmin` utility, using the `flush-hosts`, `flush-logs`, `reload` or `flush-tables` commands.

7.19 KILL syntax

```
KILL thread_id
```

Each connection to `mysqld` runs in a separate thread. You can see which threads are running with the `SHOW PROCESSLIST` command, and kill a thread with the `KILL thread_id` command.

If you have the **process** privilege, you can see and kill all threads. Otherwise, you can see and kill only your own threads.

You can also use the `mysqladmin processlist` and `mysqladmin kill` commands to examine and kill threads.

7.20 SHOW syntax (Get information about tables, columns,...)

```
   SHOW DATABASES [LIKE wild]
or SHOW TABLES [FROM db_name] [LIKE wild]
or SHOW COLUMNS FROM tbl_name [FROM db_name] [LIKE wild]
or SHOW INDEX FROM tbl_name [FROM db_name]
or SHOW STATUS
or SHOW VARIABLES [LIKE wild]
or SHOW PROCESSLIST
or SHOW TABLE STATUS [FROM db_name] [LIKE wild]
```

`SHOW` provides information about databases, tables, columns or the server. If the `LIKE wild` part is used, the `wild` string can be a string that uses the SQL '`%`' and '`_`' wildcard characters.

You can use `db_name.tbl_name` as an alternative to the `tbl_name FROM db_name` syntax. These two statements are equivalent:

```
mysql> SHOW INDEX FROM mytable FROM mydb;
mysql> SHOW INDEX FROM mydb.mytable;
```

`SHOW DATABASES` lists the databases on the **MySQL** server host. You can also get this list using the `mysqlshow` command.

`SHOW TABLES` lists the tables in a given database. You can also get this list using the `mysqlshow db_name` command.

Note: If a user doesn't have any privileges for a table, the table will not show up in the output from `SHOW TABLES` or `mysqlshow db_name`.

`SHOW COLUMNS` lists the columns in a given table. If the column types are different than you expect them to be based on a `CREATE TABLE` statement, note that MySQL sometimes changes column types. See Section 7.6.1 [Silent column changes], page 172.

The `DESCRIBE` statement provides information similar to `SHOW COLUMNS`. See Section 7.22 [DESCRIBE], page 198.

`SHOW TABLE STATUS` (new in version 3.23) works likes `SHOW STATUS`, but provides a lot of information about each table. You can also get this list using the `mysqlshow --status db_name` command. The following columns are returned:

Name	Name of the table
Type	Type of table (NISAM, MyISAM or HEAP)

Rows	Number of rows
Avg_row_length	Average row length
Data_length	Length of the data file
Max_data_length	Max length of the data file
Index_length	Length of the index file
Data_free	Number of allocated but not used bytes
Auto_increment	Next autoincrement value
Create_time	When the table was created
Update_time	When the data file was last updated
Check_time	When one last run a check on the table
Create_options	Extra options used with `CREATE TABLE`
Comment	The comment used when creating the table (or some information why MySQL couldn't access the table information).

`SHOW FIELDS` is a synonym for `SHOW COLUMNS` and `SHOW KEYS` is a synonym for `SHOW INDEX`. You can also list a table's columns or indexes with `mysqlshow db_name tbl_name` or `mysqlshow -k db_name tbl_name`.

`SHOW INDEX` returns the index information in a format that closely resembles the `SQLStatistics` call in ODBC. The following columns are returned:

Table	Name of the table
Non_unique	0 if the index can't contain duplicates.
Key_name	Name of the index
Seq_in_index	Column sequence number in index, starting with 1.
Column_name	Column name.
Collation	How the column is sorted in the index. In **MySQL**, this can have values `A` (Ascending) or `NULL` (Not sorted).
Cardinality	Number of unique values in the index. This is updated by running `isamchk -a`.
Sub_part	Number of indexed characters if the column is only partly indexed. `NULL` if the entire key is indexed.

`SHOW STATUS` provides server status information (like `mysqladmin extended-status`). The output resembles that shown below, though the format and numbers may differ somewhat:

```
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| Aborted_clients          | 0      |
| Aborted_connects         | 0      |
| Created_tmp_tables       | 0      |
| Delayed_insert_threads   | 0      |
| Delayed_writes           | 0      |
| Delayed_errors           | 0      |
| Flush_commands           | 2      |
| Handler_delete           | 2      |
| Handler_read_first       | 0      |
| Handler_read_key         | 1      |
```

```
| Handler_read_next         | 0      |
| Handler_read_rnd          | 35     |
| Handler_update            | 0      |
| Handler_write             | 2      |
| Key_blocks_used           | 0      |
| Key_read_requests         | 0      |
| Key_reads                 | 0      |
| Key_write_requests        | 0      |
| Key_writes                | 0      |
| Max_used_connections      | 1      |
| Not_flushed_key_blocks    | 0      |
| Not_flushed_delayed_rows  | 0      |
| Open_tables               | 1      |
| Open_files                | 2      |
| Open_streams              | 0      |
| Opened_tables             | 11     |
| Questions                 | 14     |
| Running_threads           | 1      |
| Slow_queries              | 0      |
| Uptime                    | 149111 |
+---------------------------+--------+
```

The status variables listed above have the following meaning:

`Aborted_clients`	Number of connections that has been aborted because the client has died without closing the connection properly.
`Aborted_connects`	Number of tries to connect to the MySQL server that has failed.
`Created_tmp_tables`	Number of implicit temporary tables that has been created while executing statements.
`Delayed_insert_threads`	Number of delayed insert handler threads in use.
`Delayed_writes`	Number of rows written with `INSERT DELAYED`.
`Delayed_errors`	Number of rows written with `INSERT DELAYED` for which some error occurred (probably `duplicate key`).
`Flush_commands`	Number of executed `FLUSH` commands.
`Handler_delete`	Number of requests to delete a row from a table.
`Handler_read_first`	Number of request to read first the row in a table.
`Handler_read_key`	Number of request to read a row based on a key.
`Handler_read_next`	Number of request to read next row in key order.
`Handler_read_rnd`	Number of request to read a row based on a fixed position.
`Handler_update`	Number of requests to update a row in a table.
`Handler_write`	Number of requests to insert a row in a table.
`Key_blocks_used`	The number of used blocks in the key cache.
`Key_read_requests`	The number of request to read a key block from the cache.
`Key_reads`	The number of physical reads of a key block from disk.
`Key_write_requests`	The number of request to write a key block to the cache.
`Key_writes`	The number of physical writes of a key block to disk.

`Max_used_connections`	The maximum number of connections that has been in use simultaneously.
`Not_flushed_key_blocks`	Keys blocks in the key cache that has changed but hasn't yet been flushed to disk.
`Not_flushed_delayed_rows`	Number of rows waiting to be written in `INSERT DELAY` queues.
`Open_tables`	Number of tables that are open.
`Open_files`	Number of files that are open.
`Open_streams`	Number of streams that are open (used mainly for logging)
`Opened_tables`	Number of tables that has been opened.
`Questions`	Number of questions asked from to the server.
`Running_threads`	Number of currently open connections.
`Slow_queries`	Number of queries that has taken more than `long_query_time`
`Uptime`	How many seconds the server has been up.

Some comments about the above:

- If `Opened_tables` is big, then your `table_cache` variable is probably too small.
- If `key_reads` is big, then your `key_cache` is probably too small. The cache hit rate can be calculated with `key_reads`/`key_read_requests`.
- If `Handler_read_rnd` is big, then you have a probably a lot of queries that requires MySQL to scan whole tables or you have joins that doesn't use keys properly.

`SHOW VARIABLES` shows the values of the some of **MySQL** system variables. You can also get this information using the `mysqladmin variables` command. If the default values are unsuitable, you can set most of these variables using command-line options when `mysqld` starts up. The output resembles that shown below, though the format and numbers may differ somewhat:

```
+-------------------------+----------------------------+
| Variable_name           | Value                      |
+-------------------------+----------------------------+
| back_log                | 5                          |
| connect_timeout         | 5                          |
| basedir                 | /my/monty/                 |
| datadir                 | /my/monty/data/            |
| delayed_insert_limit    | 100                        |
| delayed_insert_timeout  | 300                        |
| delayed_queue_size      | 1000                       |
| join_buffer_size        | 131072                     |
| flush_time              | 0                          |
| key_buffer_size         | 1048540                    |
| language                | /my/monty/share/english/   |
| log                     | OFF                        |
| log_update              | OFF                        |
| long_query_time         | 10                         |
| low_priority_updates    | OFF                        |
```

```
| max_allowed_packet     | 1048576                |
| max_connections        | 100                    |
| max_connect_errors     | 10                     |
| max_delayed_threads    | 20                     |
| max_heap_table_size    | 16777216               |
| max_join_size          | 4294967295             |
| max_sort_length        | 1024                   |
| max_tmp_tables         | 32                     |
| net_buffer_length      | 16384                  |
| port                   | 3306                   |
| protocol-version       | 10                     |
| record_buffer          | 131072                 |
| skip_locking           | ON                     |
| socket                 | /tmp/mysql.sock        |
| sort_buffer            | 2097116                |
| table_cache            | 64                     |
| thread_stack           | 131072                 |
| tmp_table_size         | 1048576                |
| tmpdir                 | /machine/tmp/          |
| version                | 3.23.0-alpha-debug     |
| wait_timeout           | 28800                  |
+------------------------+------------------------+
```

See Section 10.1 [Server parameters], page 244.

`SHOW PROCESSLIST` shows you which threads are running. You can also get this information using the `mysqladmin processlist` command. If you have the **process** privilege, you can see all threads. Otherwise, you can see only your own threads. See Section 7.19 [KILL], page 190.

7.21 EXPLAIN syntax (Get information about a SELECT)

```
EXPLAIN SELECT select_options
```

When you precede a `SELECT` statement with the keyword `EXPLAIN`, **MySQL** explains how it would process the `SELECT`, providing information about how tables are joined and in which order.

With the help of `EXPLAIN`, you can see when you must add indexes to tables to get a faster `SELECT` that uses indexes to find the records. You can also see if the optimizer joins the tables in an optimal order. To force the optimizer to use a specific join order for a `SELECT` statement, add a `STRAIGHT_JOIN` clause.

For non-simple joins, `EXPLAIN` returns a row of information for each table used in the `SELECT` statement. The tables are listed in the order they would be read. **MySQL** resolves all joins using a single-sweep multi-join method. This means that **MySQL** reads a row from the first table, then finds a matching row in the second table, then in the third table and so on. When all tables are processed, it outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

Output from `EXPLAIN` includes the following columns:

`table` The table to which the row of output refers.

`type` The join type. Information about the various types is given below.

`possible_keys`
The `possible_keys` column indicates which indexes **MySQL** could use to find the rows in the table. If this column is empty, there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the `WHERE` clause to see if it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with `EXPLAIN` again.

To see what indexes a table has, use `SHOW INDEX FROM tbl_name`.

`key` The `key` column indicates the key that **MySQL** actually decided to use. The key is `NULL` if no index was chosen.

`key_len` The `key_len` column indicates the length of the key that **MySQL** decided to use. The length is `NULL` if the `key` is `NULL`.

`ref` The `ref` column shows which columns or constants are used with the `key` to select rows from the table.

`rows` The `rows` column indicates the number of rows **MySQL** must examine to execute the query.

`Extra` If the `Extra` column includes the text `Only index`, this means that information is retrieved from the table using only information in the index tree. Normally, this is much faster than scanning the entire table.

If the `Extra` column includes the text `where used`, it means that a `WHERE` clause will be used to restrict which rows will be matched against the next table or sent to the client.

The different join types are listed below, ordered from best to worst type:

`system` The table has only one row (= system table). This is a special case of the `const` join type.

`const` The table has at most one matching row, which will be read at the start of the query. Since there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. `const` tables are very fast as they are read only once!

`eq_ref` One row will be read from this table for each combination of rows from the previous tables. This the best possible join type, other than the `const` types. It is used when all parts of an index are used by the join and the index is `UNIQUE` or a `PRIMARY KEY`.

`ref` All rows with matching index values will be read from this table for each combination of rows from the previous tables. `ref` is used if the join uses only a leftmost prefix of the key, or if the key is not `UNIQUE` or a `PRIMARY KEY` (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this join type is good.

`range` Only rows that are in a given range will be retrieved, using an index to select the rows. The `ref` column indicates which index is used.

`index` This is the same as `ALL`, except that only the index tree is scanned. This is usually faster than `ALL`, as the index file is usually smaller than the data file.

`ALL` A full table scan will be done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked `const`, and usually **very** bad in all other cases. You normally can avoid `ALL` by adding more indexes, so that the row can be retrieved based on constant values or column values from earlier tables.

You can get a good indication of how good a join is by multiplying all values in the `rows` column of the `EXPLAIN` output. This should tell you roughly how many rows **MySQL** must examine to execute the query. This number is also used when you restrict queries with the `max_join_size` variable. See Section 10.1 [Server parameters], page 244.

The following example shows how a `JOIN` can be optimized progressively using the information provided by `EXPLAIN`.

Suppose you have the `SELECT` statement shown below, that you examine using `EXPLAIN`:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
            tt.ProjectReference, tt.EstimatedShipDate,
            tt.ActualShipDate, tt.ClientID,
            tt.ServiceCodes, tt.RepetitiveID,
            tt.CurrentProcess, tt.CurrentDPPerson,
            tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
            et_1.COUNTRY, do.CUSTNAME
        FROM tt, et, et AS et_1, do
        WHERE tt.SubmitTime IS NULL
            AND tt.ActualPC = et.EMPLOYID
            AND tt.AssignedPC = et_1.EMPLOYID
            AND tt.ClientID = do.CUSTNMBR;
```

For this example, assume that:

- The columns being compared have been declared as follows:

Table	Column	Column type
tt	ActualPC	CHAR(10)
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- The tables have the indexes shown below:

Table	Index
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The `tt.ActualPC` values aren't evenly distributed.

Initially, before any optimizations have been performed, the `EXPLAIN` statement produces the following information:

```
table type possible_keys                key  key_len ref  rows  Extra
et    ALL  PRIMARY                      NULL NULL    NULL 74
do    ALL  PRIMARY                      NULL NULL    NULL 2135
et_1  ALL  PRIMARY                      NULL NULL    NULL 74
tt    ALL  AssignedPC,ClientID,ActualPC NULL NULL    NULL 3872
      range checked for each record (key map: 35)
```

Since `type` is `ALL` for each table, this output indicates that **MySQL** is doing a full join for all tables! This will take quite a long time, as the product of the number of rows in each table must be examined! For the case at hand, this is `74 * 2135 * 74 * 3872 = 45,268,558,720` rows. If the tables were bigger, you can only imagine how long it would take...

One problem here is that **MySQL** can't (yet) use indexes on columns efficiently if they are declared differently. In this context, `VARCHAR` and `CHAR` are the same unless they are declared as different lengths. Since `tt.ActualPC` is declared as `CHAR(10)` and `et.EMPLOYID` is declared as `CHAR(15)`, there is a length mismatch.

To fix this disparity between column lengths, use `ALTER TABLE` to lengthen `ActualPC` from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now `tt.ActualPC` and `et.EMPLOYID` are both `VARCHAR(15)`. Executing the `EXPLAIN` statement again produces this result:

```
table type   possible_keys   key     key_len ref         rows    Extra
tt    ALL    AssignedPC,ClientID,ActualPC NULL NULL NULL 3872    where used
do    ALL    PRIMARY         NULL    NULL    NULL        2135
      range checked for each record (key map: 1)
et_1  ALL    PRIMARY         NULL    NULL    NULL        74
      range checked for each record (key map: 1)
et    eq_ref PRIMARY         PRIMARY 15      tt.ActualPC 1
```

This is not perfect, but is much better (the product of the `rows` values is now less by a factor of 74). This version is executed in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the `tt.AssignedPC = et_1.EMPLOYID` and `tt.ClientID = do.CUSTNMBR` comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
                      MODIFY ClientID   VARCHAR(15);
```

Now `EXPLAIN` produces the output shown below:

```
table type   possible_keys   key     key_len ref            rows     Extra
et    ALL    PRIMARY         NULL    NULL    NULL           74
tt    ref    AssignedPC,ClientID,ActualPC ActualPC 15 et.EMPLOYID 52 where used
et_1  eq_ref PRIMARY         PRIMARY 15      tt.AssignedPC  1
do    eq_ref PRIMARY         PRIMARY 15      tt.ClientID    1
```

This is "almost" as good as it can get.

The remaining problem is that, by default, **MySQL** assumes that values in the `tt.ActualPC` column are evenly distributed, and that isn't the case for the `tt` table. Fortunately, it is easy to tell **MySQL** about this:

```
shell> isamchk --analyze PATH_TO_MYSQL_DATABASE/tt
shell> mysqladmin refresh
```

Now the join is "perfect", and `EXPLAIN` produces this result:

```
table type   possible_keys   key      key_len ref             rows    Extra
tt    ALL    AssignedPC,ClientID,ActualPC NULL NULL NULL     3872    where used
et    eq_ref PRIMARY         PRIMARY 15      tt.ActualPC     1
et_1  eq_ref PRIMARY         PRIMARY 15      tt.AssignedPC   1
do    eq_ref PRIMARY         PRIMARY 15      tt.ClientID     1
```

Note that the `rows` column in the output from `EXPLAIN` is an "educated guess" from the **MySQL** join optimizer; To optimize a query, you should check if the numbers are even close to the truth. If not, you may get better performance by using `STRAIGHT_JOIN` in your `SELECT` statement and trying to list the tables in a different order in the `FROM` clause.

7.22 `DESCRIBE` syntax (Get information about columns)

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

`DESCRIBE` provides information about a table's columns. `col_name` may be a column name or a string containing the SQL '`%`' and '`_`' wildcard characters.

If the column types are different than you expect them to be based on a `CREATE TABLE` statement, note that MySQL sometimes changes column types. See Section 7.6.1 [Silent column changes], page 172.

This statement is provided for Oracle compatibility.

The `SHOW` statement provides similar information. See Section 7.20 [SHOW], page 190.

7.23 `LOCK TABLES/UNLOCK TABLES` syntax

```
LOCK TABLES tbl_name [AS alias] {READ | [LOW_PRIORITY] WRITE}
            [, tbl_name {READ | [LOW_PRIORITY] WRITE} ...]
...
UNLOCK TABLES
```

`LOCK TABLES` locks tables for the current thread. `UNLOCK TABLES` releases any locks held by the current thread. All tables that are locked by the current thread are automatically unlocked when the thread issues another `LOCK TABLES`, or when the connection to the server is closed.

If a thread obtains a `READ` lock on a table, that thread (and all other threads) can only read from the table. If a thread obtains a `WRITE` lock on a table, then only the thread holding the lock can `READ` from or `WRITE` to the table. Other threads are blocked.

Each thread waits (without timing out) until it obtains all the locks it has requested.

`WRITE` locks normally have higher priority than `READ` locks, to ensure that updates are processed as soon as possible. This means that if one thread obtains a `READ` lock and then another thread requests a `WRITE` lock, subsequent `READ` lock requests will wait until the

`WRITE` thread has gotten the lock and released it. You can use `LOW_PRIORITY WRITE` locks to allow other threads to obtain `READ` locks while the thread is waiting for the `WRITE` lock. You should only use `LOW_PRIORITY WRITE` locks if you are sure that there will eventually be a time when no threads will have a `READ` lock.

When you use `LOCK TABLES`, you must lock all tables that you are going to use! If you are using a table multiple times in a query (with aliases), you must get a lock for each alias! This policy ensures that table locking is deadlock free.

Note that you should **NOT** lock any tables that you are using with `INSERT DELAYED`. This is because that in this case the `INSERT` is done by a separate thread.

Normally, you don't have to lock tables, as all single `UPDATE` statements are atomic; no other thread can interfere with any other currently executing SQL statement. There are a few cases when you would like to lock tables anyway:

- If you are going to run many operations on a bunch of tables, it's much faster to lock the tables you are going to use. The downside is, of course, that no other thread can update a `READ`-locked table and no other thread can read a `WRITE`-locked table.
- **MySQL** doesn't support a transaction environment, so you must use `LOCK TABLES` if you want to ensure that no other thread comes between a `SELECT` and an `UPDATE`. The example shown below requires `LOCK TABLES` in order to execute safely:

```
mysql> LOCK TABLES trans READ, customer WRITE;
mysql> select sum(value) from trans where customer_id= some_id;
mysql> update customer set total_value=sum_from_previous_statement
           where customer_id=some_id;
mysql> UNLOCK TABLES;
```

  Without `LOCK TABLES`, there is a chance that another thread might insert a new row in the `trans` table between execution of the `SELECT` and `UPDATE` statements.

By using incremental updates (`UPDATE customer SET value=value+new_value`) or the `LAST_INSERT_ID()` function, you can avoid using `LOCK TABLES` in many cases.

You can also solve some cases by using the user-level lock functions `GET_LOCK()` and `RELEASE_LOCK()`. These locks are saved in a hash table in the server and implemented with `pthread_mutex_lock()` and `pthread_mutex_unlock()` for high speed. See Section 7.3.12 [Miscellaneous functions], page 162.

See Section 10.11 [Internal locking], page 255, for more information on locking policy.

7.24 `SET OPTION` syntax

```
SET [OPTION] SQL_VALUE_OPTION= value, ...
```

`SET OPTION` sets various options that affect the operation of the server or your client. Any option you set remains in effect until the current session ends, or until you set the option to a different value.

`CHARACTER SET character_set_name | DEFAULT`

This maps all strings from and to the client with the given mapping. Currently the only option for `character_set_name` is `cp1251_koi8`, but you can easily add new mappings by editing the '`sql/convert.cc`' file in the **MySQL** source

distribution. The default mapping can be restored by using a `character_set_name` value of `DEFAULT`.

Note that the syntax for setting the `CHARACTER SET` option differs from the syntax for setting the other options.

`PASSWORD = PASSWORD('some password')`
Set the password for the current user. Any non-anonymous user can change his own password!

`PASSWORD FOR user = PASSWORD('some password')`
Set the password for a specific user on the current server host. Only a user with access to the `mysql` database can do this. The user should be given in `user@hostname` format, where `user` and `hostname` are exactly as they are listed in the `User` and `Host` columns of the `mysql.user` table entry. For example, if you had an entry with `User` and `Host` fields of `'bob'` and `'%.loc.gov'`, you would write:

```
mysql> SET PASSWORD FOR bob@"%.loc.gov" = PASSWORD("newpass");
```

`SQL_BIG_TABLES = 0 | 1`
If set to `1`, all temporary tables are stored on disk rather than in memory. This will be a little slower, but you will not get the error `The table tbl_name is full` for big `SELECT` operations that require a large temporary table. The default value for a new connection is `0` (i.e., use in-memory temporary tables).

`SQL_BIG_SELECTS = 0 | 1`
If set to `1`, **MySQL** will abort if a `SELECT` is attempted that probably will take a very long time. This is useful when an inadvisable `WHERE` statement has been issued. A big query is defined as a `SELECT` that probably will have to examine more than `max_join_size` rows. The default value for a new connection is `0` (which will allow all `SELECT` statements).

`SQL_LOW_PRIORITY_UPDATES = 0 | 1`
If set to `1`, all `INSERT`, `UPDATE` and `DELETE` statements wait until there is no pending `SELECT` on the affected table.

`SQL_SELECT_LIMIT = value | DEFAULT`
The maximum number of records to return from `SELECT` statements. If a `SELECT` has a `LIMIT` clause, the `LIMIT` takes precedence over the value of `SQL_SELECT_LIMIT`. The default value for a new connection is "unlimited". If you have changed the limit, the default value can be restored by using a `SQL_SELECT_LIMIT` value of `DEFAULT`.

`SQL_LOG_OFF = 0 | 1`
If set to `1`, no logging will be done to the standard log for this client, if the client has the **process** privilege. This does not affect the update log!

`SQL_LOG_UPDATE = 0 | 1`
If set to `0`, no logging will be done to the update log for the client, if the client has the **process** privilege. This does not affect the standard log!

`TIMESTAMP = timestamp_value | DEFAULT`
: Set the time for this client. This is used to get the original timestamp if you use the update log to restore rows.

`LAST_INSERT_ID = #`
: Set the value to be returned from `LAST_INSERT_ID()`. This is stored in the update log when you use `LAST_INSERT_ID()` in a command that updates a table.

`INSERT_ID = #`
: Set the value to be used by the following `INSERT` command when inserting an `AUTO_INCREMENT` value. This is mainly used with the update log.

7.25 `GRANT` and `REVOKE` syntax

```
GRANT priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {tbl_name | * | *.* | db_name.*}
    TO user_name [IDENTIFIED BY 'password']
        [, user_name [IDENTIFIED BY 'password'] ...]
    [WITH GRANT OPTION]

REVOKE priv_type [(column_list)] [, priv_type [(column_list)] ...]
    ON {tbl_name | * | *.* | db_name.*}
    FROM user_name [, user_name ...]
```

`GRANT` is implemented in **MySQL** 3.22.11 or later. For earlier **MySQL** versions, the `GRANT` statement does nothing.

The `GRANT` and `REVOKE` commands allow system administrators to grant and revoke rights to **MySQL** users at four privilege levels:

Global level
: Global privileges apply to all databases on a given server. These privileges are stored in the `mysql.user` table.

Database level
: Database privileges apply to all tables in a given database. These privileges are stored in the `mysql.db` and `mysql.host` tables.

Table level
: Table privileges apply to all columns in a given table. These privileges are stored in the `mysql.tables_priv` table.

Column level
: Column privileges apply to single columns in a given table. These privileges are stored in the `mysql.columns_priv` table.

For examples of how `GRANT` works, see Section 6.11 [Adding users], page 107.

For the `GRANT` and `REVOKE` statements, `priv_type` may be specified as any of the following:

```
ALL PRIVILEGES      FILE                RELOAD
ALTER               INDEX               SELECT
```

```
CREATE              INSERT              SHUTDOWN
DELETE              PROCESS             UPDATE
DROP                REFERENCES          USAGE
```

`ALL` is a synonym for `ALL PRIVILEGES`. `REFERENCES` is not yet implemented. `USAGE` is currently a synonym for "no privileges". It can be used when you want to create a user that has no privileges.

To revoke the **grant** privilege from a user, use a `priv_type` value of `GRANT OPTION`:

```
REVOKE GRANT OPTION ON ... FROM ...;
```

The only `priv_type` values you can specify for a table are `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `CREATE`, `DROP`, `GRANT`, `INDEX` and `ALTER`.

The only `priv_type` values you can specify for a column (that is, when you use a `column_list` clause) are `SELECT`, `INSERT` and `UPDATE`.

You can set global privileges by using `ON *.*` syntax. You can set database privileges by using `ON db_name.*` syntax. If you specify `ON *` and you have a current database, you will set the privileges for that database. (**Warning:** If you specify `ON *` and you *don't* have a current database, you will affect the global privileges!)

In order to accommodate granting rights to users from arbitrary hosts, **MySQL** supports specifying the `user_name` value in the form `user@host`. If you want to specify a `user` string containing special characters (such as '`-`'), or a `host` string containing special characters or wildcard characters (such as '`%`'), you can quote the user or host name (e.g., `'test-user'@'test-hostname'`).

You can specify wildcards in the hostname. For example, `user@"%.loc.gov"` applies to `user` for any host in the `loc.gov` domain, and `user@"144.155.166.%"` applies to `user` for any host in the `144.155.166` class C subnet.

The simple form `user` is a synonym for `user@"%"`. **Note:** If you allow anonymous users to connect to the **MySQL** server (which is the default), you should also add all local users as `user@localhost` because otherwise the anonymous user entry for the local host in the `mysql.user` table will be used when the user tries to log into the **MySQL** server from the local machine! Anonymous users are defined by inserting entries with `User=''` into the `mysql.user` table. You can verify if this applies to you by executing this query:

```
mysql> SELECT Host,User FROM mysql.user WHERE User='';
```

For the moment, `GRANT` only supports host, table, database and column names up to 60 characters long. A user name can be up to 16 characters.

The privileges for a table or column are formed from the logical OR of the privileges at each of the four privilege levels. For example, if the `mysql.user` table specifies that a user has a global **select** privilege, this can't be denied by an entry at the database, table or column level.

The privileges for a column can be calculated as follows:

```
global privileges
OR (database privileges AND host privileges)
OR table privileges
OR column privileges
```

In most cases, you grant rights to a user at only one of the privilege levels, so life isn't normally as complicated as above. :) The details of the privilege-checking procedure are presented in Chapter 6 [Privilege system], page 94.

If you grant privileges for a user/hostname combination that does not exist in the `mysql.user` table, an entry is added and remains there until deleted with a `DELETE` command. In other words, `GRANT` may create `user` table entries, but `REVOKE` will not remove them; you must do that explicitly using `DELETE`.

In **MySQL** 3.22.12 or later, if a new user is created or if you have global grant privileges, the user's password will be set to the password specified by the `IDENTIFIED BY` clause, if one is given. If the user already had a password, it is replaced by the new one.

Warning: If you create a new user but do not specify an `IDENTIFIED BY` clause, the user has no password. This is insecure.

Passwords can also be set with the `SET PASSWORD` command. See Section 7.24 [SET OPTION], page 199.

If you grant privileges for a database, an entry in the `mysql.db` table is created if needed. When all privileges for the database have been removed with `REVOKE`, this entry is deleted.

If a user doesn't have any privileges on a table, the table is not displayed when the user requests a list of tables (e.g., with a `SHOW TABLES` statement).

The `WITH GRANT OPTION` clause gives the user the ability to give to other users any privileges the user has at the specified privilege level. You should be careful to whom you give the **grant** privilege, as two users with different privileges may be able to join privileges!

You cannot grant another user a privilege you don't have yourself; the **grant** privilege allows you to give away only those privileges you possess.

Be aware that when you grant a user the **grant** privilege at a particular privilege level, any privileges the user already possesses (or is given in the future!) at that level are also grantable by that user. Suppose you grant a user the **insert** privilege on a database. If you then grant the **select** privilege on the database and specify `WITH GRANT OPTION`, the user can give away not only the **select** privilege, but also **insert**. If you then grant the **update** privilege to the user on the database, the user can give away the **insert**, **select** and **update**.

You should not grant **alter** privileges to a normal user. If you do that, the user can try to subvert the privilege system by renaming tables!

Note that if you are using table or column privileges for even one user, the server examines table and column privileges for all users and this will slow down **MySQL** a bit.

When `mysqld` starts, all privileges are read into memory. Database, table and column privileges take effect at once and user-level privileges take effect the next time the user connects. Modifications to the grant tables that you perform using `GRANT` or `REVOKE` are noticed by the server immediately. If you modify the grant tables manually (using `INSERT`, `UPDATE`, etc.), you should execute a `FLUSH PRIVILEGES` statement or run `mysqladmin flush-privileges` to tell the server to reload the grant tables. See Section 6.9 [Privilege changes], page 106.

The biggest differences between the ANSI SQL and **MySQL** versions of `GRANT` are:

- ANSI SQL doesn't have global or database-level privileges and ANSI SQL doesn't support all privilege types that **MySQL** supports.

- When you drop a table in ANSI SQL, all privileges for the table are revoked. If you revoke a privilege in ANSI SQL, all privileges that were granted based on this privilege are also revoked. In **MySQL**, privileges can be dropped only with explicit `REVOKE` commands or by manipulating the **MySQL** grant tables.

7.26 `CREATE INDEX` syntax

```
CREATE [UNIQUE] INDEX index_name ON tbl_name (col_name[(length)],... )
```

The `CREATE INDEX` statement doesn't do anything in **MySQL** prior to version 3.22. In 3.22 or later, `CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See Section 7.7 [`ALTER TABLE`], page 172.

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See Section 7.6 [`CREATE TABLE`], page 167. `CREATE INDEX` allows you to add indexes to existing tables.

A column list of the form `(col1,col2,...)` creates a multiple-column index. Index values are formed by concatenating the values of the given columns.

For `CHAR` and `VARCHAR` columns, indexes can be created that use only part of a column, using `col_name(length)` syntax. (On `BLOB` and `TEXT` columns the length is required). The statement shown below creates an index using the first 10 characters of the `name` column:

```
mysql> CREATE INDEX part_of_name ON customer (name(10));
```

Since most names usually differ in the first 10 characters, this index should not be much slower than an index created from the entire `name` column. Also, using partial columns for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up `INSERT` operations!

Note that you can only add a index on a column that can have `NULL` values or on a `BLOB`/`TEXT` column if you are useing **MySQL** version 3.23.2 or newer and are using the `MyISAM` table type.

For more information about how **MySQL** uses indexes, see Section 10.4 [**MySQL** indexes], page 249.

7.27 `DROP INDEX` syntax

```
DROP INDEX index_name
```

`DROP INDEX` doesn't do anything in **MySQL** prior to version 3.22. In 3.22 or later, `DROP INDEX` is mapped to an `ALTER TABLE` statement to drop the index. See Section 7.7 [`ALTER TABLE`], page 172.

7.28 Comment syntax

The **MySQL** server supports the `# to end of line`, `-- to end of line` and `/* in-line or multiple-line */` comment styles:

```
mysql> select 1+1;     # This comment continues to the end of line
mysql> select 1+1;     -- This comment continues to the end of line
mysql> select 1 /* this is an in-line comment */ + 1;
mysql> select 1+
/*
this is a
multiple-line comment
*/
1;
```

Note that the `--` comment style requires you to have at least one space after the `--`!

Although the server understands the comment syntax just described, there are some limitations on the way that the `mysql` client parses `/* ... */` comments:

- Single-quote and double-quote characters are taken to indicate the beginning of a quoted string, even within a comment. If the quote is not matched by a second quote within the comment, the parser doesn't realize the comment has ended. If you are running `mysql` interactively, you can tell that it has gotten confused like this because the prompt changes from `mysql>` to `'>` or `">`.
- A semicolon is taken to indicate the end of the current SQL statement and anything following it to indicate the beginning of the next statement.

These limitations apply both when you run `mysql` interactively and when you put commands in a file and tell `mysql` to read its input from that file with `mysql < some-file`.

MySQL doesn't support the '`--`' ANSI SQL comment style. See Section 5.3.7 [Missing comments], page 90.

7.29 `CREATE FUNCTION/DROP FUNCTION` syntax

```
CREATE FUNCTION function_name RETURNS {STRING|REAL|INTEGER}
       SONAME shared_library_name

DROP FUNCTION function_name
```

A user-definable function (UDF) is a way to extend **MySQL** with a new function that works like native (built in) **MySQL** functions such as `ABS()` and `CONCAT()`.

`CREATE FUNCTION` saves the function's name, type and shared library name in the `mysql.func` system table. You must have the **insert** and **delete** privileges for the `mysql` database to create and drop functions.

All active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable. (An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`.)

For instructions on writing user-definable functions, see Chapter 14 [Adding functions], page 294. For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading.

7.30 Is MySQL picky about reserved words?

A common problem stems from trying to create a table with column names that use the names of datatypes or functions built into **MySQL**, such as `TIMESTAMP` or `GROUP`. You're allowed to do it (for example, `ABS` is an allowed column name), but whitespace is not allowed between a function name and the '`(`' when using functions whose names are also column names.

The following words are explicitly reserved in **MySQL**. Most of them are forbidden by ANSI SQL92 as column and/or table names (for example, `group`). A few are reserved because **MySQL** needs them and is (currently) using a `yacc` parser:

action	add	all	alter
after	and	as	asc
auto_increment	between	bigint	bit
binary	blob	bool	both
by	cascade	char	character
change	check	column	columns
constraint	create	cross	current_date
current_time	current_ timestamp	data	database
databases	date	datetime	day
day_hour	day_minute	day_second	dayofmonth
dayofweek	dayofyear	dec	decimal
default	delete	desc	describe
distinct	distinctrow	double	drop
escaped	enclosed	enum	explain
exists	fields	first	float
float4	float8	foreign	from
for	full	function	grant
group	having	hour	hour_minute
hour_second	ignore	in	index
infile	insert	int	integer
interval	int1	int2	int3
int4	int8	into	if
is	join	key	keys
last_insert_id	leading	left	like
lines	limit	load	lock
long	longblob	longtext	low_priority
match	mediumblob	mediumtext	mediumint
middleint	minute	minute_second	month
monthname	natural	numeric	no
not	null	on	option
optionally	or	order	outer
outfile	partial	password	precision
primary	procedure	processlist	privileges

quarter	read	real	references
rename	regexp	reverse	repeat
replace	restrict	returns	rlike
second	select	set	show
smallint	soname	sql_big_tables	sql_big_selects
sql_select_limit	sql_low_priority_updates	sql_log_off	sql_log_update
straight_join	starting	status	string
table	tables	terminated	text
time	timestamp	tinyblob	tinytext
tinyint	trailing	to	use
using	unique	unlock	unsigned
update	usage	values	varchar
variables	varying	varbinary	with
write	where	year	year_month
zerofill			

The following symbols (from the table above) are disallowed by ANSI SQL but allowed by **MySQL** as column/table names. This is because some of these names are very natural names and a lot of people have already used them.

- ACTION
- BIT
- DATE
- ENUM
- NO
- TEXT
- TIME
- TIMESTAMP

8 MySQL Tutorial

This chapter provides a tutorial introduction to **MySQL** by showing how to use the `mysql` client program to create and use a simple database. `mysql` (sometimes referred to as the "terminal monitor" or just "monitor") is an interactive program that allows you to connect to a **MySQL** server, run queries and view the results. `mysql` may also be used in batch mode: you place your queries in a file beforehand, then tell `mysql` to execute the contents of the file. Both ways of using `mysql` are covered here.

To see a list of options provided by `mysql`, invoke it with the `--help` option:

```
shell> mysql --help
```

This chapter assumes that `mysql` is installed on your machine, and that a **MySQL** server is available to which you can connect. If this is not true, contact your **MySQL** administrator. (If *you* are the administrator, you will need to consult other sections of this manual.)

The chapter describes the entire process of setting up and using a database. If you are interested only in accessing an already-existing database, you may want to skip over the sections that describe how to create the database and the tables it contains.

Since this chapter is tutorial in nature, many details are necessarily left out. Consult the relevant sections of the manual for more information on the topics covered here.

8.1 Connecting to and disconnecting from the server

To connect to the server, you'll usually need to provide a **MySQL** user name when you invoke `mysql` and, most likely, a password. If the server runs on a machine other than the one where you log in, you'll also need to specify a hostname. Contact your administrator to find out what connection parameters you should use to connect (i.e., what host, user name and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
Enter password: ********
```

The `********` represents your password; enter it when `mysql` displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 459 to server version: 3.22.20a-log

Type 'help' for help.

mysql>
```

The prompt tells you that `mysql` is ready for you to enter commands.

Some **MySQL** installations allow users to connect as the "anonymous" (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking `mysql` without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing `QUIT` at the `mysql>` prompt:

```
mysql> QUIT
Bye
```

You can also disconnect by typing control-D.

Most examples in the following sections assume you are connected to the server. They indicate this by the `mysql>` prompt.

8.2 Entering queries

Make sure you are connected to the server, as discussed in the previous section. Doing so will not in itself select any database to work with, but that's okay. At this point, it's more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them and retrieving data from them. This section describes the basic principles of entering commands, using several queries you can try out to familiarize yourself with how `mysql` works.

Here's a simple command that asks the server to tell you its version number and the current date. Type it in as shown below following the `mysql>` prompt and hit the RETURN key:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
| version()    | CURRENT_DATE |
+--------------+--------------+
| 3.22.20a-log | 1999-03-19   |
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about `mysql`:

- A command normally consists of a SQL statement followed by a semicolon. (There are some exceptions where a semicolon is not needed. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a command, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` to indicate that it is ready for another command.
- `mysql` displays query output as a table (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), `mysql` labels the column using the expression itself.
- `mysql` shows how many rows were returned, and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the "rows in set" line is not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here's another query. It demonstrates that you can use `mysql` as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-------------+---------+
|    0.707107 |      25 |
+-------------+---------+
```

The commands shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+--------------+
| version()    |
+--------------+
| 3.22.20a-log |
+--------------+

+---------------------+
| NOW()               |
+---------------------+
| 1999-03-19 00:15:33 |
+---------------------+
```

A command need not be given all on a single line, so lengthy commands that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here's a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+--------------------+--------------+
| USER()             | CURRENT_DATE |
+--------------------+--------------+
| joesmith@localhost | 1999-03-18   |
+--------------------+--------------+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it hasn't seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you will always be aware of what `mysql` is waiting for.

If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new command.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in:

Prompt	Meaning
`mysql>`	Ready for new command
`->`	Waiting for next line of multiple-line command
`'>`	Waiting for next line, collecting a string that begins with a single quote (“'”)
`">`	Waiting for next line, collecting a string that begins with a double quote (“"”)

Multiple-line statements commonly occur “by accident” when you intend to issue a command on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```
mysql> SELECT USER()
    ->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` will execute it:

```
mysql> SELECT USER()
    -> ;
+--------------------+
| USER()             |
+--------------------+
| joesmith@localhost |
+--------------------+
```

The `'>` and `">` prompts occur during string collection. In **MySQL**, you can write strings surrounded by either ‘'’ or ‘"’ characters (for example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When you see a `'>` or `">` prompt, it means that you've entered a line containing a string that begins with a ‘'’ or ‘"’ quote character, but have not yet entered the matching quote that terminates the string. That's fine if you really are entering a multiple-line string, but how likely is that? Not very. More often, the `'>` and `">` prompts indicate that you've inadvertantly left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;
    ">
```

If you enter this `SELECT` statement, then hit RETURN and wait for the result, nothing will happen. Instead of wondering, “why does this query take so long?,” notice the clue provided by the `">` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `"Smith` is missing the second quote.)

At this point, what do you do? The simplest thing is to cancel the command. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that

it is collecting! Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = "Smith AND age < 30;
    "> "\c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new command.

It's important to know what the `'>` and `">` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type will appear to be ignored by `mysql` — including a line containing `QUIT`! This can be quite confusing, especially if you don't know that you need to supply the terminating quote before you can cancel the current command.

8.3 Examples of common queries

Here follows examples of how to solve some common problems with **MySQL**.

Some of the examples uses the table 'shop' holds the prices of each article (item number) for certain traders. Supposing that each trader has a single fixed price per article, then (item, trader) is a primary key for the records.

You can create the example table as:

```
CREATE TABLE shop (
 article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
 dealer  CHAR(20)                 DEFAULT ''     NOT NULL,
 price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL,
 PRIMARY KEY(article, dealer));

INSERT INTO shop VALUES
(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),(3,'C',1.69),
(3,'D',1.25),(4,'D',19.95);
```

Okay, so the example data is:

```
SELECT * FROM shop

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | A      |  3.45 |
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | B      |  1.45 |
|    0003 | C      |  1.69 |
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

8.3.1 The maximum value for column

"What's the highest item number?"

```
SELECT MAX(article) AS article FROM shop

+---------+
| article |
+---------+
|       4 |
+---------+
```

8.3.2 The row holding the maximum of a certain column

"Find number, dealer, and price of the most expensive article"

In ANSI-SQL this is easily done with a sub-query:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop)
```

In **MySQL** (not having sub-selects yet) just do it in two steps:

1. Get the maximum value from the table with a `SELECT` statment.
2. Using this value compile the actual query:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=19.95
```

Another solution is to sort all rows descending by price and only get the first row using the **MySQL** specific `LIMIT` clause:

```
SELECT article, dealer, price
FROM   shop
ORDER BY price DESC
LIMIT 1
```

Note: If there are several most expensive articles (e.g. each 19.95) the `LIMIT` solution does only show one of them!

8.3.3 Maximum of column: per group: only the values

"What's the highest price per article?"

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

8.3.4 The rows holding the group-wise maximum of a certain field

"For each article find the dealer(s) with the most expensive price."

In ANSI SQL I'd do it with a sub-query like this:

```
SELECT article, dealer, price
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article)
```

In **MySQL** it's best do it in several steps:

1. Get the list of (article,maxprice). See Section 8.3.4 [example-Maximum-column-group-row], page 214.
2. For each article get the corresponding rows which have the stored maximum price.

This can easily be done with a temporary table:

```
CREATE TEMPORARY TABLE tmp (
        article INT(4) UNSIGNED ZEROFILL DEFAULT '0000' NOT NULL,
        price   DOUBLE(16,2)             DEFAULT '0.00' NOT NULL);

LOCK TABLES article read;

INSERT INTO tmp SELECT article, MAX(price) FROM shop GROUP BY article;

SELECT article, dealer, price FROM shop, tmp
WHERE shop.article=tmp.articel AND shop.price=tmp.price;

UNLOCK TABLES;

DROP TABLE tmp;
```

If you don't use a TEMPORARY table, you must also lock the 'tmp' table.

"Can it be done with a single query?"

Yes, but only by using a quite inefficient trick that I call the "MAX-CONCAT trick":

```
SELECT article,
       SUBSTRING( MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 7) AS dealer,
  0.00+LEFT(      MAX( CONCAT(LPAD(price,6,'0'),dealer) ), 6) AS price
FROM   shop
GROUP BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

The last example can of course be made a bit more efficient by doing the splitting of the concatenated column in the client.

8.3.5 Using foreign keys

One doesn't need foreign keys to join 2 tables.

The only thing `MySQL` doesn't do is `CHECK` to make sure that the keys you use really exist in the table(s) you're referencing and it doesn't automatically delete rows from table with a foreign key definition. If you use your keys like normal, and it'll work just fine!

```
CREATE TABLE persons (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirts (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES persons,
    PRIMARY KEY (id)
);


INSERT INTO persons VALUES (NULL, 'Antonio Paz');

INSERT INTO shirts VALUES
(NULL, 'polo', 'blue', LAST_INSERT_ID()),
(NULL, 'dress', 'white', LAST_INSERT_ID()),
(NULL, 't-shirt', 'blue', LAST_INSERT_ID());


INSERT INTO persons VALUES (NULL, 'Lilliana Angelovska');

INSERT INTO shirts VALUES
(NULL, 'dress', 'orange', LAST_INSERT_ID()),
(NULL, 'polo', 'red', LAST_INSERT_ID()),
(NULL, 'dress', 'blue', LAST_INSERT_ID()),
(NULL, 't-shirt', 'white', LAST_INSERT_ID());


SELECT * FROM persons;
+----+--------------------+
| id | name               |
+----+--------------------+
|  1 | Antonio Paz        |
|  2 | Lilliana Angelovska |
+----+--------------------+
```

```
SELECT * FROM shirts;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+


SELECT s.* FROM persons p, shirts s
 WHERE p.name LIKE 'Lilliana%'
   AND s.owner = p.id
   AND s.color &lt;&gt; 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

8.4 Creating and using a database

Now that you know how to enter commands, it's time to access a database.

Suppose you have several pets in your home (your "menagerie") and you'd like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows how to do all that:

- How to create a database
- How to create a table
- How to load data into the table
- How to retrieve data from the table in various ways
- How to use multiple tables

The menagerie database will be simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records.

Use the `SHOW` statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

The list of databases is probably different on your machine, but the `mysql` and `test` databases are likely to be among them. The `mysql` database is required because it describes user access privileges. The `test` database is often provided as a workspace for users to try things out.

If the `test` database exists, try to access it:

```
mysql> USE test
Database changed
```

Note that `USE`, like `QUIT`, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The `USE` statement is special in another way, too: it must be given on a single line.

You can use the `test` database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your **MySQL** administrator for permission to use a database of your own. Suppose you want to call yours `menagerie`. The administrator needs to execute a command like this:

```
mysql> GRANT ALL ON menagerie.* TO your_mysql_name;
```

where `your_mysql_name` is the **MySQL** user name assigned to you.

8.4.1 Creating and selecting a database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case sensitive (unlike SQL keywords), so you must always refer to your database as `menagerie`, not as `Menagerie`, `MENAGERIE` or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query.)

Creating a database does not select it for use, you must do that explicitly. To make `menagerie` the current database, use this command:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a `mysql` session. You can do this by issuing a `USE` statement as shown above. Alternatively, you can select the database on the command line when you invoke `mysql`. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Note that `menagerie` is not your password on the command just shown. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (e.g., as `-pmypassword`, not as `-p mypassword`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

8.4.2 Creating a table

Creating the database is the easy part, but at this point it's empty, as `SHOW TABLES` will tell you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you will need, and what columns will be in each of them.

You'll want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Since the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it's not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it's better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. **MySQL** provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you'll soon need to send out birthday greetings, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient for now: name, owner, species, sex, birth and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
    -> species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner` and `species` columns since the column values will vary in length. The lengths of those columns need not all be the same, and need not

be 20. You can pick any length from 1 to 255, whatever seems most reasonable to you. (If you make a poor choice and it turns out later that you need a longer field, **MySQL** provides an ALTER TABLE statement.)

Animal sex can be represented in a variety of ways, for example, "m" and "f", or perhaps "male" and "female". It's simplest to use the single characters "m" and "f".

The use of the DATE data type for the birth and death columns is a fairly obvious choice.

Now that you have created a table, SHOW TABLES should produce some output:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| pet                 |
+---------------------+
```

To verify that your table was created the way you expected, use a DESCRIBE statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use DESCRIBE any time, for example, if you forget the names of the columns in your table or what types they are.

8.4.3 Loading data into a table

After creating your table, you need to populate it. The LOAD DATA and INSERT statements are useful for this.

Suppose your pet records can be described as shown below. (Observe that **MySQL** expects dates in YYYY-MM-DD format; this may be different than what you are used to.)

name	**owner**	**species**	**sex**	**birth**	**death**
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1998-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Since you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file '`pet.txt`' containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing values (such as unknown sexes, or death dates for animals that are still living), you can use `NULL` values. To represent these in your text file, use `\N`. For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler      Gwen        bird        \N          1997-12-09          \N
```

To load the text file '`pet.txt`' into the `pet` table, use this command:

```
mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

You can specify the column value separator and end of line marker explicitly in the `LOAD DATA` statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file '`pet.txt`' properly.

When you want to add new records one at a time, the `INSERT` statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the `CREATE TABLE` statement. Suppose Diane gets a new hamster named Puffball. You could add a new record using an `INSERT` statement like this:

```
mysql> INSERT INTO pet
    -> VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

Note that string and date values are specified as quoted strings here. Also, with `INSERT`, you can insert `NULL` directly to represent a missing value. You do not use `\N` like you do with `LOAD DATA`.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several `INSERT` statements rather than a single `LOAD DATA` statement.

8.4.4 Retrieving information from a table

The `SELECT` statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy
```

`what_to_select` indicates what you want to see. This can be a list of columns, or `*` to to indicate "all columns." `which_table` indicates the table from which you want to retrieve data. The `WHERE` clause is optional. If it's present, `conditions_to_satisfy` specifies conditions that rows must satisfy to qualify for retrieval.

8.4.4.1 Selecting all data

The simplest form of `SELECT` retrieves everything from a table:

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1998-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

This form of `SELECT` is useful if you want to review your entire table, for instance, after you've just loaded it with your initial dataset. As it happens, the output just shown reveals an error in your data file: Bowser appears to have been born after he died! Consulting your original pedigree papers, you find that the correct birth year is 1989, not 1998.

There are are least a couple of ways to fix this:

- Edit the file '`pet.txt`' to correct the error, then empty the table and reload it using `DELETE` and `LOAD DATA`:

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
  ```

  However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an `UPDATE` statement:

  ```
  mysql> UPDATE pet SET birth = "1989-08-31" WHERE name = "Bowser";
  ```

As shown above, it is easy to retrieve an entire table. But typically you don't want to do that, particularly when the table becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

8.4.4.2 Selecting particular rows

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = "Bowser";
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded now as 1989, not 1998.

String comparisons are normally case-insensitive, so you can specify the name as `"bowser"`, `"BOWSER"`, etc. The query result will be the same.

You can specify conditions on any column, not just `name`. For example, if you want to know which animals were born after 1998, test the `birth` column:

```
mysql> SELECT * FROM pet WHERE birth >= "1998-1-1";
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = "dog" AND sex = "f";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The preceding query uses the `AND` logical operator. There is also an `OR` operator:

```
mysql> SELECT * FROM pet WHERE species = "snake" OR species = "bird";
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

`AND` and `OR` may be intermixed. If you do that, it's a good idea to use parentheses to indicate how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = "cat" AND sex = "m")
    -> OR (species = "dog" AND sex = "f");
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

8.4.4.3 Selecting particular columns

If you don't want to see entire rows from your table, just name the columns in which you're interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
```

```
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

However, notice that the query simply retrieves the `owner` field from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
    -> WHERE species = "dog" OR species = "cat";
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
```

```
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

8.4.4.4 Sorting rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. However, it's often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To sort in reverse order, add the `DESC` (descending) keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
+----------+------------+
```

You can sort on multiple columns. For example, to sort by type of animal, then by birth date within animal type with youngest animals first, use the following query:

```
mysql> SELECT name, species, birth FROM pet ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
```

```
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

Note that the `DESC` keyword applies only to the column name immediately preceding it (`birth`); `species` values are still sorted in ascending order.

8.4.4.5 Date calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, compute age as the difference between the birth date and the current date. Do this by converting the two dates to days, take the difference, and divide by 365 (the number of days in a year):

```
mysql> SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 FROM pet;
+----------+-------------------------------------+
| name     | (TO_DAYS(NOW())-TO_DAYS(birth))/365 |
+----------+-------------------------------------+
| Fluffy   |                                6.15 |
| Claws    |                                5.04 |
| Buffy    |                                9.88 |
| Fang     |                                8.59 |
| Bowser   |                                9.58 |
| Chirpy   |                                0.55 |
| Whistler |                                1.30 |
| Slim     |                                2.92 |
| Puffball |                                0.00 |
+----------+-------------------------------------+
```

Although the query works, there are some things about it that could be improved. First, the result could be scanned more easily if the rows were presented in some order. Second, the heading for the age column isn't very meaningful.

The first problem can be handled by adding an `ORDER BY name` clause to sort the output by name. To deal with the column heading, provide a name for the column so that a different label appears in the output (this is called a column alias):

```
mysql> SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 AS age
    -> FROM pet ORDER BY name;
+----------+------+
| name     | age  |
+----------+------+
| Bowser   | 9.58 |
| Buffy    | 9.88 |
| Chirpy   | 0.55 |
```

```
| Claws    | 5.04 |
| Fang     | 8.59 |
| Fluffy   | 6.15 |
| Puffball | 0.00 |
| Slim     | 2.92 |
| Whistler | 1.30 |
+----------+------+
```

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql>  SELECT name, (TO_DAYS(NOW())-TO_DAYS(birth))/365 AS age
    ->  FROM pet ORDER BY age;
+----------+------+
| name     | age  |
+----------+------+
| Puffball | 0.00 |
| Chirpy   | 0.55 |
| Whistler | 1.30 |
| Slim     | 2.92 |
| Claws    | 5.04 |
| Fluffy   | 6.15 |
| Fang     | 8.59 |
| Bowser   | 9.58 |
| Buffy    | 9.88 |
+----------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether or not the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql>  SELECT name, birth, death, (TO_DAYS(death)-TO_DAYS(birth))/365 AS age
    ->  FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 | 5.91 |
+--------+------------+------------+------+
```

The query uses `death IS NOT NULL` rather than `death != NULL` because `NULL` is a special value. This is explained later. See Section 8.4.4.6 [Working with NULL], page 227.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant, you simply want to extract the month part of the `birth` column. **MySQL** provides several date-part extraction functions, such as `YEAR()`, `MONTH()` and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)`:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
```

```
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

Finding animals with birthdays in the upcoming month is easy, too. Suppose the current month is April. Then the month value is `4` and you look for animals born in May (month 5) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
+-------+------------+
```

There is a small complication if the current month is December, of course. You don't just add one to the month number (`12`) and look for animals born in month 13, because there is no such month. Instead, you look for animals born in January (month 1).

You can even write the query so that it works no matter what the current month is. That way you don't have to use a particular month number in the query. `DATE_ADD()` allows you to add a time interval to a given date. If you add a month to the value of `NOW()`, then extract the month part with `MONTH()`, the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MONTH(DATE_ADD(NOW(), INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add `1` to get the next month after the current one (after using the modulo function (`MOD`) to "wrap around" the month value to `0` if it is currently `12`):

```
mysql> SELECT name, birth FROM pet
    -> WHERE MONTH(birth) = MOD(MONTH(NOW()),12) + 1;
```

8.4.4.6 Working with `NULL` values

The `NULL` value can be surprising until you get used to it. Conceptually, `NULL` means "missing value" or "unknown value" and it is treated somewhat differently than other values. To test for `NULL`, you cannot use the arithmetic comparison operators such as `=`, `<` or `!=`. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 != NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 != NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
```

```
+----------+-----------+----------+----------+
```

Clearly you get no meaningful results from these comparisons. Use the `IS NULL` and `IS NOT NULL` operators instead:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

In **MySQL**, 0 means false and 1 means true.

This special treatment of `NULL` is why, in the previous section, it was necessary to determine which animals are no longer alive using `death IS NOT NULL` instead of `death != NULL`.

8.4.4.7 Pattern matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as `vi`, `grep` and `sed`.

SQL pattern matching allows you to use '`_`' to match any single character, and '`%`' to match an arbitrary number of characters (including zero characters). SQL patterns are case insensitive. Some examples are shown below. Note that you do not use `=` or `!=` when you use SQL patterns; use the `LIKE` or `NOT LIKE` comparison operators instead.

To find names beginning with '`b`':

```
mysql> SELECT * FROM pet WHERE name LIKE "b%";
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with '`fy`':

```
mysql> SELECT * FROM pet WHERE name LIKE "%fy";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a '`w`':

```
mysql> SELECT * FROM pet WHERE name LIKE "%w%";
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
```

```
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

To find names containing exactly five characters, use the '_' pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE "_____";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The other type of pattern matching provided by **MySQL** uses extended regular expressions. When you test for a match for this type of pattern, use the `REGEXP` and `NOT REGEXP` operators (or `RLIKE` and `NOT RLIKE`, which are synonyms).

Some characteristics of extended regular expressions are:

- '`.`' matches any single character.
- A character class '`[...]`' matches any character within the brackets. For example, '`[abc]`' matches '`a`', '`b`' or '`c`'. To name a range of characters, use a dash. '`[a-z]`' matches any lowercase letter, whereas '`[0-9]`' matches any digit.
- '`*`' matches zero or more instances of the thing preceding it. For example, '`x*`' matches any number of '`x`' characters, '`[0-9]*`' matches any number of digits, and '`.*`' matches any number of anything.
- Regular expressions are case sensitive, but you can use a character class to match both lettercases if you wish. For example, '`[aA]`' matches lowercase or uppercase '`a`' and '`[a-zA-Z]`' matches any letter in either case.
- The pattern matches if it occurs anywhere in the value being tested (SQL patterns match only if they match the entire value).
- To anchor a pattern so that it must match the beginning or end of the value being tested, use '`^`' at the beginning or '`$`' at the end of the pattern.

To demonstrate how extended regular expressions work, the `LIKE` queries shown above are rewritten below to use `REGEXP`:

To find names beginning with '`b`', use '`^`' to match the beginning of the name and '`[bB]`' to match either lowercase or uppercase '`b`':

```
mysql> SELECT * FROM pet WHERE name REGEXP "^[bB]";
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with '`fy`', use '`$`' to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP "fy$";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
```

```
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a 'w', use '`[wW]`' to match either lowercase or uppercase '`w`':

```
mysql> SELECT * FROM pet WHERE name REGEXP "[wW]";
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Since a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value like it would be if you used an SQL pattern.

To find names containing exactly five characters, use '`^`' and '`$`' to match the beginning and end of the name, and five instances of '`.`' in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP "^.....$";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

You could also write the previous query using the '`{n}`' "repeat-`n`-times" operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP "^.{5}$";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

8.4.4.8 Counting rows

Databases are often used to answer the question, "how often does a certain type of data occur in a table?" For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of censuses on your animals.

Counting the total number of animals you have is the same question as "how many rows are in the `pet` table?," since there is one record per pet. The `COUNT()` function counts the number of non-`NULL` results, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
```

```
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
+--------+----------+
```

Note the use of `GROUP BY` to group together all records for each `owner`. Without it, all you get is an error message:

```
mysql> SELECT owner, COUNT(owner) FROM pet;
ERROR 1140 at line 1: Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

`COUNT()` and `GROUP BY` are useful for characterizing your data in various ways. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(In this output, `NULL` indicates "sex unknown.")

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use `COUNT()`. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE species = "dog" OR species = "cat"
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

Or, if you wanted the number of animals per sex only for known-sex animals:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
    -> WHERE sex IS NOT NULL
    -> GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

8.4.5 Using more than one table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like?

- It needs to contain the pet name so you know which animal each event pertains to.
- It needs a date so you know when the event occurred.
- It needs a field to describe the event.
- If you want to be able to categorize events, it would be useful to have an event type field.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
    -> type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it's easiest to load the initial records by creating a tab-delimited text file containing the information:

Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE "event.txt" INTO TABLE event;
```

Based on what you've learned from the queries you've run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose you want to find out the ages of each pet when they had their litters. The `event` table indicates when this occurred, but to calculate age of the mother, you need her birth date. Since that is stored in the `pet` table, you need both tables for the query:

```
mysql> SELECT pet.name, (TO_DAYS(date) - TO_DAYS(birth))/365 AS age, remark
    -> FROM pet, event
    -> WHERE pet.name = event.name AND type = "litter";
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy | 2.27 | 4 kittens, 3 female, 1 male |
| Buffy  | 4.12 | 5 puppies, 2 female, 3 male |
| Buffy  | 5.10 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

There are several things to note about this query:

- The `FROM` clause lists two tables since the query needs to pull information from both of them.

- When combining information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy since they both have a `name` column. The query uses `WHERE` clause to match up records in the two tables based on the `name` values.
- Since the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the `pet` table with itself to pair up males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
    -> FROM pet AS p1, pet AS p2
    -> WHERE p1.species = p2.species AND p1.sex = "f" AND p2.sex = "m";
+--------+------+--------+------+---------+
| name   | sex  | name   | sex  | species |
+--------+------+--------+------+---------+
| Fluffy | f    | Claws  | m    | cat     |
| Buffy  | f    | Fang   | m    | dog     |
| Buffy  | f    | Bowser | m    | dog     |
+--------+------+--------+------+---------+
```

In this query, we specify aliases for the table name in order to be able to refer to the columns and keep straight which instance of the table each column reference is associated with.

8.5 Getting information about databases and tables

What if you forget the name of a database or table, or what the structure of a given table is (e.g., what its columns are called)? **MySQL** addresses this problem through several statements that provide information about the databases and tables it supports.

You have already seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` function:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

If you haven't selected any database yet, the result is blank.

To find out what tables the current database contains (for example, when you're not sure about the name of a table), use this command:

```
mysql> SHOW TABLES;
+---------------------+
| Tables in menagerie |
+---------------------+
| event               |
```

```
| pet                  |
+----------------------+
```

If you want to find out about the structure of a table, the `DESCRIBE` command is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

`Field` indicates the column name, `Type` is the data type for the column, `Null` indicates whether or not the column can contain `NULL` values, `Key` indicates whether or not the column is indexed and `Default` specifies the column's default value.

If you have indexes on a table, `SHOW INDEX FROM tbl_name` produces information about them.

8.6 Using `mysql` in batch mode

In the previous sections, you used `mysql` interactively to enter queries and view the results. You can also run `mysql` in batch mode. To do this, put the commands you want to run in a file, then tell `mysql` to read its input from the file:

```
shell> mysql < batch-file
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

When you use `mysql` this way, you are creating a script file, then executing the script.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script allows you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line commands or multiple-statement sequences of commands. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

  ```
  shell> mysql < batch-file | more
  ```

- You can catch the output in a file for further processing:

```
shell> mysql < batch-file > mysql.out
```

- You can distribute your script to other people so they can run the commands, too.
- Some situations do not allow for interactive use, for example, when you run a query from a `cron` job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when run interactively:

```
+---------+
| species |
+---------+
| bird    |
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

But like this when run in batch mode:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the commands that are executed, use `mysql -vvv`.

8.7 Queries from twin project

At Analytikerna and Lentus, we have been doing the systems and field work for a big research project. This project is a collaboration between the Institute of Environmental Medicine at Karolinska Institutet Stockholm and the Section on Clinical Research in Aging and Psychology at the University of Southern California.

The project involves a screening part where all twins in Sweden older than 65 years are interviewed by telephone. Twins who meet certain criteria are passed on to the next stage. In this latter stage, twins who want to participate are visited by a doctor/nurse team. Some of the examinations include physical and neuropsychological examination, laboratory testing, neuroimaging, psychological status assessment, and family history collection. In addition, data are collected on medical and environmental risk factors.

More information about Twin studies can be found at:

```
http://www.imm.ki.se/TWIN/TWINUKW.HTM
```

The latter part of the project is administered with a web interface written using Perl and **MySQL**.

Each night all data from the interviews are moved into a **MySQL** database.

8.7.1 Find all non-distributed twins

The following query is used to determine who goes into the second part of the project:

```
select
        concat(p1.id, p1.tvab) + 0 as tvid,
        concat(p1.christian_name, " ", p1.surname) as Name,
        p1.postal_code as Code,
        p1.city as City,
        pg.abrev as Area,
        if(td.participation = "Aborted", "A", " ") as A,
        p1.dead as dead1,
        l.event as event1,
        td.suspect as tsuspect1,
        id.suspect as isuspect1,
        td.severe as tsevere1,
        id.severe as isevere1,
        p2.dead as dead2,
        l2.event as event2,
        h2.nurse as nurse2,
        h2.doctor as doctor2,
        td2.suspect as tsuspect2,
        id2.suspect as isuspect2,
        td2.severe as tsevere2,
        id2.severe as isevere2,
        l.finish_date
from
        twin_project as tp
        /* For Twin 1 */
        left join twin_data as td on tp.id = td.id and tp.tvab = td.tvab
        left join informant_data as id on tp.id = id.id and tp.tvab = id.tvab
        left join harmony as h on tp.id = h.id and tp.tvab = h.tvab
        left join lentus as l on tp.id = l.id and tp.tvab = l.tvab
        /* For Twin 2 */
        left join twin_data as td2 on p2.id = td2.id and p2.tvab = td2.tvab
        left join informant_data as id2 on p2.id = id2.id and p2.tvab = id2.tvab
        left join harmony as h2 on p2.id = h2.id and p2.tvab = h2.tvab
        left join lentus as l2 on p2.id = l2.id and p2.tvab = l2.tvab,
        person_data as p1,
        person_data as p2,
        postal_groups as pg
where
        /* p1 gets main twin and p2 gets his/her twin. */
        /* ptvab is a field inverted from tvab */
        p1.id = tp.id and p1.tvab = tp.tvab and
        p2.id = p1.id and p2.ptvab = p1.tvab and
        /* Just the sceening survey */
        tp.survey_no = 5 and
        /* Skip if partner died before 65 but allow emigration (dead=9) */
```

```
            (p2.dead = 0 or p2.dead = 9 or
             (p2.dead = 1 and
              (p2.death_date = 0 or
               (((to_days(p2.death_date) - to_days(p2.birthday)) / 365)
                >= 65))))
            and
            (
            /* Twin is suspect */
            (td.future_contact = 'Yes' and td.suspect = 2) or
            /* Twin is suspect - Informant is Blessed */
            (td.future_contact = 'Yes' and td.suspect = 1 and id.suspect = 1) or
            /* No twin - Informant is Blessed */
            (ISNULL(td.suspect) and id.suspect = 1 and id.future_contact = 'Yes') or
            /* Twin broken off - Informant is Blessed */
            (td.participation = 'Aborted'
             and id.suspect = 1 and id.future_contact = 'Yes') or
            /* Twin broken off - No inform - Have partner */
            (td.participation = 'Aborted' and ISNULL(id.suspect) and p2.dead = 0))
            and
            l.event = 'Finished'
            /* Get at area code */
            and substring(p1.postal_code, 1, 2) = pg.code
            /* Not already distributed */
            and (h.nurse is NULL or h.nurse=00 or h.doctor=00)
            /* Has not refused or been aborted */
            and not (h.status = 'Refused' or h.status = 'Aborted'
            or h.status = 'Died' or h.status = 'Other')
    order by
            tvid;
```

Some explanations:

`concat(p1.id, p1.tvab) + 0 as tvid`
: We want to sort on the concatenated `id` and `tvab` in numerical order. Adding `0` to the result causes **MySQL** to treat the result as a number.

column `id`
: This identifies a pair of twins. It is a key in all tables.

column `tvab`
: This identifies a twin in a pair. It has a value of `1` or `2`.

column `ptvab`
: This is an inverse of `tvab`. When `tvab` is `1` this is `2`, and vice versa. It exists to save typing and to make it easier for **MySQL** to optimize the query.

This query demonstrates, among other things, how to do lookups on a table from the same table with a join (`p1` and `p2`). In the example, this is used to check whether a twin's partner died before the age of 65. If so, the row is not returned.

All of the above exist in all tables with twin-related information. We have a key on both `id,tvab` (all tables) and `id,ptvab` (`person_data`) to make queries faster.

On our production machine (A 200MHz UltraSPARC), this query returns about 150-200 rows and takes less than one second.

The current number of records in the tables used above:

Table	Rows
person_data	71074
lentus	5291
twin_project	5286
twin_data	2012
informant_data	663
harmony	381
postal_groups	100

8.7.2 Show a table on twin pair status

Each interview ends with a status code called `event`. The query shown below is used to display a table over all twin pairs combined by event. This indicates in how many pairs both twins are finished, in how many pairs one twin is finished and the other refused, and so on.

```
select
        t1.event,
        t2.event,
        count(*)
from
        lentus as t1,
        lentus as t2,
        twin_project as tp
where
        /* We are looking at one pair at a time */
        t1.id = tp.id
        and t1.tvab=tp.tvab
        and t1.id = t2.id
        /* Just the sceening survey */
        and tp.survey_no = 5
        /* This makes each pair only appear once */
        and t1.tvab='1' and t2.tvab='2'
group by
        t1.event, t2.event;
```

9 MySQL server functions

9.1 What languages are supported by MySQL?

`mysqld` can issue error messages in the following languages: Czech, Dutch, English (the default), Estonia, French, German, Hungarian, Italian, Norwegian, Norwegian-ny, Polish, Portuguese, Spanish and Swedish.

To start `mysqld` with a particular language, use either the `--language=lang` or `-L lang` options. For example:

```
shell> mysqld --language=swedish
```

or:

```
shell> mysqld --language=/usr/local/share/swedish
```

Note that all language names are specified in lowercase.

The language files are located (by default) in '*mysql_base_dir*`/share/`*LANGUAGE*`/`'.

To update the error message file, you should edit the '`errmsg.txt`' file and execute the following command to generate the '`errmsg.sys`' file:

```
shell> comp_err errmsg.txt errmsg.sys
```

If you upgrade to a newer version of **MySQL**, remember to repeat your changes with the new '`errmsg.txt`' file.

9.1.1 The character set used for data and sorting

By default, **MySQL** uses the ISO-8859-1 (Latin1) character set. This is the character set used in the USA and western Europe.

The character set determines what characters are allowed in names and how things are sorted by the `ORDER BY` and `GROUP BY` clauses of the `SELECT` statement.

You can change the character set at compile time by using the `--with-charset=charset` option to `configure`. See Section 4.7.1 [Quick install], page 40.

To add another character set to **MySQL**, use the following procedure:

9.1.2 Adding a new character set

1. Choose a name for the character set, denoted `MYSET` below.
2. Create the file '`strings/ctype-MYSET.c`' in the **MySQL** source distribution.
3. Look at one of the existing '`ctype-*.c`' files to see what needs to be defined. Note that the arrays in your file must have names like `ctype_MYSET`, `to_lower_MYSET` and so on.

   `to_lower[]` and `to_upper[]` are simple arrays that hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

   ```
   to_lower['A'] should contain 'a'
   to_upper['a'] should contain 'A'
   ```

`sort_order[]` is a map indicating how characters should be ordered for comparison and sorting purposes. For many character sets, this is the same as `to_upper[]` (which means sorting will be case insensitive). **MySQL** will sort characters based on the value of `sort_order[character]`.

`ctype[]` is an array of bit values, with one element for one character. (Note that `to_lower[]`, `to_upper[]` and `sort_order[]` are indexed by character value, but `ctype[]` is indexed by character value + 1. This is an old legacy to be able to handle EOF.) You can find the following bitmask definitions in '`m_ctype.h`':

```
#define _U      01      /* Upper case */
#define _L      02      /* Lower case */
#define _N      04      /* Numeral (digit) */
#define _S      010     /* Spacing character */
#define _P      020     /* Punctuation */
#define _C      040     /* Control character */
#define _B      0100    /* Blank */
#define _X      0200    /* heXadecimal digit */
```

The `ctype[]` entry for each character should be the union of the applicable bitmask values that describe the character. For example, `'A'` is an uppercase character (`_U`) as well as a hexadecimal digit (`_X`), so `ctype['A'+1]` should contain the value:

```
_U + _X = 01 + 0200 = 0201
```

4. Add a unique number for your character set to '`include/m_ctype.h.in`'.
5. Add the character set name to the `CHARSETS_AVAILABLE` list in `configure.in`.
6. Reconfigure, recompile and test.

9.1.3 Multi-byte character support

If you are creating a multi-byte character set, you can use the `_MB` macros. In '`include/m_ctype.h.in`', add:

```
#define MY_CHARSET_MYSET  X
#if MY_CHARSET_CURRENT == MY_CHARSET_MYSET
#define USE_MB
#define USE_MB_IDENT
#define ismbchar(p, end)  (...)
#define ismbhead(c)       (...)
#define mbcharlen(c)      (...)
#define MBMAXLEN          N
#endif
```

Where:

`MY_CHARSET_MYSET`	A unique character set value.
`USE_MB`	This character set has multi-byte characters, handled by `ismbhead()` and `mbcharlen()`
`USE_MB_IDENT`	(optional) If defined, you can use table and column names that use multi-byte characters

`ismbchar(p, e)`	return 0 if `p` is not a multi-byte character string, or the size of the character (in bytes) if it is. `p` and `e` point to the beginning and end of the string. Check from `(char*)p` to `(char*)e-1`.
`ismbhead(c)`	True if `c` is the first character of a multi-byte character string
`mbcharlen(c)`	Size of a multi-byte character string if `c` is the first character of such a string
`MBMAXLEN`	Size in bytes of the largest character in the set

9.2 The update log

When started with the `--log-update=file_name` option, `mysqld` writes a log file containing all SQL commands that update data. The file is written in the data directory and has a name of `file_name.#`, where `#` is a number that is incremented each time you execute `mysqladmin refresh` or `mysqladmin flush-logs`, the `FLUSH LOGS` statement, or restart the server.

If you use the `--log` or `-l` options, the filename is '`hostname.log`', and restarts and refreshes do not cause a new log file to be generated. By default, the `mysql.server` script starts the **MySQL** server with the `-l` option. If you need better performance when you start using **MySQL** in a production environment, you can remove the `-l` option from `mysql.server`.

Update logging is smart since it logs only statements that really update data. So an `UPDATE` or a `DELETE` with a `WHERE` that finds no rows is not written to the log. It even skips `UPDATE` statements that set a column to the value it already has.

If you want to update a database from update log files, you could do the following (assuming your log files have names of the form '`file_name.#`'):

```
shell> ls -1 -t -r file_name.[0-9]* | xargs cat | mysql
```

`ls` is used to get all the log files in the right order.

This can be useful if you have to revert to backup files after a crash and you want to redo the updates that occurred between the time of the backup and the crash.

You can also use the update logs when you have a mirrored database on another host and you want to replicate the changes that have been made to the master database.

9.3 How big MySQL tables can be

MySQL itself has a 4G limit on table size, and operating systems have their own file size limits. On Linux, the current limit is 2G; on Solaris 2.5.1, the limit is 4G; on Solaris 2.6, the limit is going to be 1000G. Currently, table sizes are limited to either 4G (the **MySQL** limit) or the operating system limit, whichever is smaller. To get more than 4G requires some changes to **MySQL** that are on the TODO. See Appendix F [TODO], page 439.

If your big table is going to be read-only, you could use `pack_isam` to merge and compress many tables to one. `pack_isam` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. See Section 12.5 [pack_isam], page 272.

Another solution can be the included MERGE library, which allows you to handle a collection of identical tables as one. (Identical in this case means that all tables are created with

identical column information.) Currently MERGE can only be used to scan a collection of tables because it doesn't support indexes. We will add indexes to this in the near future.

10 Getting maximum performance from MySQL

10.1 Tuning server parameters

You can get the default buffer sizes used by the `mysqld` server with this command:

```
shell> mysqld --help
```

This command produces a list of all `mysqld` options and configurable variables. The output includes the default values and looks something like this:

```
Possible variables for option --set-variable (-O) are:
back_log              current value: 5
connect_timeout       current value: 5
delayed_insert_timeout  current value: 300
delayed_insert_limit  current value: 100
delayed_queue_size    current value: 1000
flush_time            current value: 0
join_buffer_size      current value: 131072
key_buffer_size       current value: 1048540
long_query_time       current value: 10
max_allowed_packet    current value: 1048576
max_connections       current value: 100
max_connect_errors    current value: 10
max_delayed_threads   current value: 20
max_heap_table_size   current value: 16777216
max_join_size         current value: 4294967295
max_sort_length       current value: 1024
max_tmp_tables        current value: 32
net_buffer_length     current value: 16384
record_buffer         current value: 131072
sort_buffer           current value: 2097116
table_cache           current value: 64
tmp_table_size        current value: 1048576
thread_stack          current value: 131072
wait_timeout          current value: 28800
```

If there is a `mysqld` server currently running, you can see what values it actually is using for the variables by executing this command:

```
shell> mysqladmin variables
```

Each option is described below. Values for buffer sizes, lengths and stack sizes are given in bytes. You can specify values with a suffix of '`K`' or '`M`' to indicate kilobytes or megabytes. For example, `16M` indicates 16 megabytes. Case of suffix letters does not matter; `16M` and `16m` are equivalent.

You can also see some statistics from a running server by the command `SHOW STATUS`. See Section 7.20 [SHOW], page 190.

`back_log` The number of outstanding connection requests **MySQL** can have. This comes into play when the main **MySQL** thread gets **VERY** many connection requests

in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The `back_log` value indicates how many requests can be stacked during this short time before **MySQL** momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix `listen(2)` system call should have more details. Check your OS documentation for the maximum value for this variable. Attempting to set `back_log` higher than this maximum will be ineffective.

`connect_timeout`
: The number of seconds the `mysqld` server is waiting for a connect packet before responding with `Bad handshake`.

`delayed_insert_timeout`
: How long a `INSERT DELAYED` thread should wait for `INSERT` statements before terminating.

`delayed_insert_limit`
: After inserting `delayed_insert_limit` rows, the `INSERT DELAYED` handler will check if there are any `SELECT` statements pending. If so, it allows these to execute before continuing.

`delayed_queue_size`
: How big a queue (in rows) should be allocated for handling `INSERT DELAYED`. If the queue becomes full, any client that does `INSERT DELAYED` will wait until there is room in the queue again.

`flush_time`
: If this is set, then every `flush_time` seconds all tables will be closed (to free up resources and sync things to disk).

`join_buffer`
: The size of the buffer that is used for full joins (joins that do not use indexes). The buffer is allocated one time for each full join between two tables. Increase this value to get a faster full join when adding indexes is not possible. (Normally the best way to get fast joins is to add indexes.)

`key_buffer`
: Index blocks are buffered and are shared by all threads. `key_buffer` is the size of the buffer used for index blocks. You might want to increase this value when doing many `DELETE` or `INSERT` operations on a table with lots of indexes. To get even more speed, use `LOCK TABLES`. See Section 7.23 [LOCK TABLES], page 198.

`long_query_time`
: If a query takes longer than this (in seconds), the `slow_queries` counter will be incremented.

`max_allowed_packet`
: The maximum size of one packet. The message buffer is initialized to `net_buffer_length` bytes, but can grow up to `max_allowed_packet` bytes when

needed. This value by default is small to catch big (possibly wrong) packets. You must increase this value if you are using big `BLOB` columns. It should be as big as the biggest `BLOB` you want to use.

`max_connections`
: The number of simultaneous clients allowed. Increasing this value increases the number of file descriptors that `mysqld` requires. See below for comments on file descriptor limits.

`max_connect_errors`
: If there is more than this number of interrupted connections from a host this host will be blocked for further connections. You can unblock a host with the command `FLUSH HOSTS`.

`max_delayed_threads Don't start more than this number of`
: threads to handle `INSERT DELAYED`. If you try to insert data in a new table after all `INSERT DELAYED` threads are in use, the row will be inserted as if the `DELAYED` attribute wasn't specified.

`max_join_size`
: Joins that are probably going to read more than `max_join_size` records return an error. Set this value if your users tend to perform joins without a `WHERE` clause that take a long time and return millions of rows.

`max_sort_length`
: The number of bytes to use when sorting `BLOB` or `TEXT` values (only the first `max_sort_length` bytes of each value are used; the rest are ignored).

`max_tmp_tables`
: (This option doesn't yet do anything). Maximum number of temporary tables a client can keep open at the same time.

`net_buffer_length`
: The communication buffer is reset to this size between queries. This should not normally be changed, but if you have very little memory, you can set it to the expected size of a query. (That is, the expected length of SQL statements sent by clients. If statements exceed this length, the buffer is automatically enlarged, up to `max_allowed_packet` bytes.)

`record_buffer`
: Each thread that does a sequential scan allocates a buffer of this size for each table it scans. If you do many sequential scans, you may want to increase this value.

`sort_buffer`
: Each thread that needs to do a sort allocates a buffer of this size. Increase this value for faster `ORDER BY` or `GROUP BY` operations. See Section 18.5 [Temporary files], page 313.

`table_cache`
: The number of open tables for all threads. Increasing this value increases the number of file descriptors that `mysqld` requires. **MySQL** needs two file de-

scriptors for each unique open table. See below for comments on file descriptor limits. For information about how the table cache works, see Section 10.8 [Table cache], page 253.

`tmp_table_size`
: If a temporary table exceeds this size, **MySQL** generates an error of the form `The table tbl_name is full`. Increase the value of `tmp_table_size` if you do many advanced `GROUP BY` queries.

`thread_stack`
: The stack size for each thread. Many of the limits detected by the `crash-me` test are dependent on this value. The default is normally large enough. See Chapter 11 [Benchmarks], page 265.

`wait_timeout`
: The number of seconds the server waits for activity on a connection before closing it.

`table_cache`, `max_connections` and `max_tmp_tables` affect the maximum number of files the server keeps open. If you increase one or both of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. However, you can increase the limit on many systems. Consult your OS documentation to find out how to do this, because the method for changing the limit varies widely from system to system.

`table_cache` is related to `max_connections`. For example, for 200 open connections, you should have a table cache of at least `200 * n`, where `n` is the maximum number of tables in a join.

MySQL uses algorithms that are very scalable, so you can usually run with very little memory or give **MySQL** more memory to get better performance.

If you have much memory and many tables and want maximum performance with a moderate number of clients, you should use something like this:

```
shell> safe_mysqld -O key_buffer=16M -O table_cache=128 \
           -O sort_buffer=4M -O record_buffer=1M &
```

If you have little memory and lots of connections, use something like this:

```
shell> safe_mysqld -O key_buffer=512k -O sort_buffer=100k \
           -O record_buffer=100k &
```

or even:

```
shell> safe_mysqld -O key_buffer=512k -O sort_buffer=16k \
           -O table_cache=32 -O record_buffer=8k -O net_buffer=1K &
```

If there are very many connections, "swapping problems" may occur unless `mysqld` has been configured to use very little memory for each connection. `mysqld` performs better if you have enough memory for all connections, of course.

Note that if you change an option to `mysqld`, it remains in effect only for that instance of the server.

To see the effects of a parameter change, do something like this:

```
shell> mysqld -O key_buffer=32m --help
```

Make sure that the `--help` option is last; otherwise, the effect of any options listed after it on the command line will not be reflected in the output.

10.2 How MySQL uses memory

The list below indicates some of the ways that the `mysqld` server uses memory. Where applicable, the name of the server variable relevant to the memory use is given.

- The key buffer (variable `key_buffer`) is shared by all threads; Other buffers used by the server are allocated as needed.
- Each connection uses some thread specific space; A stack (64K, variable `thread_stack`) a connection buffer (variable `net_buffer_length`), and a result buffer (variable `net_buffer_length`). The connection buffer and result buffer are dynamically enlarged up to `max_allowed_packet` when needed. When a query is running a copy of the current query string is also allocated.
- All threads share the same base memory.
- Nothing is memory-mapped yet (except compressed tables, but that's another story). This is because the 32-bit memory space of 4GB is not large enough for most large tables. When we get a system with a 64-bit address space, we may add general support for memory-mapping.
- Each request doing a sequential scan over a table allocates a read buffer (variable `record_buffer`).
- All joins are done in one pass and most joins can be done without even using a temporary table. Most temporary tables are memory-based (HEAP) tables. Temporary tables with a big record length (calculated as the sum of all column lengths) or that contain `BLOB` columns are stored on disk. One current problem is that if a HEAP table exceeds the size of `tmp_table_size`, you get the error `The table tbl_name is full`. In the future, we will fix this by automatically changing the in-memory (HEAP) table to a disk-based (NISAM) table as necessary. To work around this problem, you can increase the temporary table size by setting the `tmp_table_size` option to `mysqld`, or by setting the SQL option `SQL_BIG_TABLES` in the client program. See Section 7.24 [SET OPTION], page 199. In **MySQL** 3.20, the maximum size of the temporary table was `record_buffer*16`, so if you are using this version, you have to increase the value of `record_buffer`. You can also start `mysqld` with the `--big-tables` option to always store temporary tables on disk, however, this will affect the speed of all complicated queries.
- Most requests doing a sort allocate a sort buffer and one or two temporary files. See Section 18.5 [Temporary files], page 313.
- Almost all parsing and calculating is done in a local memory store. No memory overhead is needed for small items and the normal slow memory allocation and freeing is avoided. Memory is allocated only for unexpectedly large strings (this is done with `malloc()` and `free()`).
- Each index file is opened once and the data file is opened once for each concurrently-running thread. For each concurrent thread, a table structure, column structures for

each column, and a buffer of size `3 * n` is allocated (where `n` is the maximum row length, not counting `BLOB` columns). A `BLOB` uses 5 to 8 bytes plus the length of the `BLOB` data.

- For each table having `BLOB` columns, a buffer is enlarged dynamically to read in larger `BLOB` values. If you scan a table, a buffer as large as the largest `BLOB` value is allocated.
- Table handlers for all in-use tables are saved in a cache and managed as a FIFO. Normally the cache has 64 entries. If a table has been used by two running threads at the same time, the cache contains two entries for the table. See Section 10.8 [Table cache], page 253.
- A `mysqladmin flush-tables` command closes all tables that are not in use and marks all in-use tables to be closed when the currently executing thread finishes. This will effectively free most in-use memory.

`ps` and other system status programs may report that `mysqld` uses a lot of memory. This may be caused by thread-stacks on different memory addresses. For example, the Solaris version of `ps` counts the unused memory between stacks as used memory. You can verify this by checking available swap with `swap -s`. We have tested `mysqld` with commercial memory-leakage detectors, so there should be no memory leaks.

10.3 How compiling and linking affects the speed of MySQL

Most of the following tests are done on Linux and with the **MySQL** benchmarks, but they should give some indication for other operating systems.

You get the fastest executable when you link with `-static`. Using Unix sockets rather than TCP/IP to connect to a database also gives better performance.

On Linux, you will get the fastest code when compiling with `pgcc` and `-O6`. To compile '`sql_yacc.cc`' with these options, you need 180M memory because `gcc/pgcc` needs a lot of memory to make all functions inline. You should also set `CXX=gcc` when configuring **MySQL** to avoid inclusion of the `libstdc++` library.

- If you use `pgcc` and compile everything with `-O6`, the `mysqld` server is 11% faster than with `gcc`.
- If you link dynamically (without `-static`), the result is 13% slower.
- If you connect using TCP/IP rather than Unix sockets, the result is 7.5% slower.
- On a Sun SPARCstation 10, `gcc` 2.7.3 is 13% faster than Sun Pro C++ 4.2.
- On Solaris 2.5.1, MIT-pthreads is 8-12% slower than Solaris native threads.

The **MySQL**-Linux distribution provided by TcX is compiled with `pgcc` and linked statically.

10.4 How MySQL uses indexes

All indexes (`PRIMARY`, `UNIQUE` and `INDEX()`) are stored in B-trees. Strings are automatically prefix- and end-space compressed. See Section 7.26 [`CREATE INDEX`], page 204.

Indexes are used to:

- Quickly find the rows that match a `WHERE` clause.

- Retrieve rows from other tables when performing joins.
- Find the `MAX()` or `MIN()` value for a specific key.
- Sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable key (e.g., `ORDER BY key_part_1,key_part_2`). The key is read in reverse order if all key parts are followed by `DESC`.
- Retrieve values without consulting the data file, in some cases. If all used columns for some table are numeric and form a leftmost prefix for some key, the values may be retrieved from the index tree for greater speed.

Suppose you issue the following `SELECT` statement:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on `col1` and `col2`, the appropriate rows can be fetched directly. If separate single-column indexes exist on `col1` and `col2`, the optimizer decides which index will find fewer rows and uses that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to find rows. For example, if you have a three-column index on `(col1,col2,col3)`, you have indexed search capabilities on `(col1)`, `(col1,col2)` and `(col1,col2,col3)`.

MySQL can't use a partial index if the columns don't form a leftmost prefix of the index. Suppose you have the `SELECT` statements shown below:

```
mysql> SELECT * FROM tbl_name WHERE col1=val1;
mysql> SELECT * FROM tbl_name WHERE col2=val2;
mysql> SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on `(col1,col2,col3)`, only the first query shown above uses the index. The second and third queries do involve indexed columns, but `(col2)` and `(col2,col3)` are not leftmost prefixes of `(col1,col2,col3)`.

MySQL also uses indexes for `LIKE` comparisons if the argument to `LIKE` is a constant string that doesn't start with a wildcard character. For example, the following `SELECT` statements use indexes:

```
mysql> select * from tbl_name where key_col LIKE "Patrick%";
mysql> select * from tbl_name where key_col LIKE "Pat%_ck%";
```

In the first statement, only rows with `"Patrick" <= key_col < "Patricl"` are considered. In the second statement, only rows with `"Pat" <= key_col < "Pau"` are considered.

The following `SELECT` statements will not use indexes:

```
mysql> select * from tbl_name where key_col LIKE "%Patrick%";
mysql> select * from tbl_name where key_col LIKE other_col;
```

In the first statement, the `LIKE` value begins with a wildcard character. In the second statement, the `LIKE` value is not a constant.

Searching using `column_name IS NULL` will use indexes if column_name is a index.

MySQL normally uses the index that finds least number of rows. An index is used for columns that you compare with the following operators: `=`, `>`, `>=`, `<`, `<=`, `BETWEEN` and a `LIKE` with a non-wildcard prefix like `'something%'`.

Any index that doesn't span all AND levels in the WHERE clause is not used to optimize the query.

The following WHERE clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2
... WHERE index=1 OR A=10 AND index=2      /* index = 1 OR index = 2 */
... WHERE index_part1='hello' AND index_part_3=5
          /* optimized like "index_part1='hello'" */
```

These WHERE clauses do **NOT** use indexes:

```
... WHERE index_part2=1 AND index_part3=2  /* index_part_1 is not used */
... WHERE index=1 OR A=10                  /* No index */
... WHERE index_part1=1 OR index_part2=10  /* No index spans all rows */
```

10.5 How MySQL optimizes WHERE clauses

(This section is incomplete; **MySQL** does many optimizations.)

In general, when you want to make a slow SELECT ... WHERE faster, the first thing to check is whether or not you can add an index. All references between different tables should usually be done with indexes. You can use the EXPLAIN command to determine which indexes are used for a SELECT. See Section 7.21 [EXPLAIN], page 194.

Some of the optimizations performed by **MySQL** are listed below:

- Removal of unnecessary parentheses:

```
   ((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
   (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal (needed because of constant folding):

```
   (B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
-> B=5 OR B=6
```

- Constant expressions used by indexes are evaluated only once.
- COUNT(*) on a single table without a WHERE is retrieved directly from the table information. This is also done for any NOT NULL expression when used with only one table.
- Early detection of invalid constant expressions. **MySQL** quickly detects that some SELECT statements are impossible and returns no rows.
- HAVING is merged with WHERE if you don't use GROUP BY or group functions (COUNT(), MIN()...)
- For each sub join, a simpler WHERE is constructed to get a fast WHERE evaluation for each sub join and also to skip records as soon as possible.
- All constant tables are read first, before any other tables in the query. A constant table is:
  - An empty table or a table with 1 row.

  - A table that is used with a `WHERE` clause on a `UNIQUE` index or a `PRIMARY KEY`, where all index parts are used with constant expressions.

  All the following tables are used as constant tables:

```
mysql> SELECT * FROM t WHERE primary_key=1;
mysql> SELECT * FROM t1,t2
           WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- The best join combination to join the tables is found by trying all possibilities :(. If all columns in `ORDER BY` and in `GROUP BY` come from the same table, then this table is preferred first when joining.
- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use `SQL_SMALL_RESULT`, **MySQL** will use an in-memory temporary table.
- As `DISTINCT` is converted to a `GROUP BY` on all columns, `DISTINCT` combined with `ORDER BY` will in many cases also need a temporary table.
- Each table index is queried and the best index that spans less than 30% of the rows is used. If no such index can be found, a quick table scan is used.
- In some cases, **MySQL** can read rows from the index without even consulting the data file. If all columns used from the index are numeric, then only the index tree is used to resolve the query.
- Before each record is output, those that do not match the `HAVING` clause are skipped.

Some examples of queries that are very fast:

```
mysql> SELECT COUNT(*) FROM tbl_name;
mysql> SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;
mysql> SELECT MAX(key_part2) FROM tbl_name
           WHERE key_part_1=constant;
mysql> SELECT ... FROM tbl_name
           ORDER BY key_part1,key_part2,... LIMIT 10;
mysql> SELECT ... FROM tbl_name
           ORDER BY key_part1 DESC,key_part2 DESC,... LIMIT 10;
```

The following queries are resolved using only the index tree (assuming the indexed columns are numeric):

```
mysql> SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;
mysql> SELECT COUNT(*) FROM tbl_name
           WHERE key_part1=val1 AND key_part2=val2;
mysql> SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
mysql> SELECT ... FROM tbl_name ORDER BY key_part1,key_part2,...
mysql> SELECT ... FROM tbl_name ORDER BY key_part1 DESC,key_part2 DESC,...
```

10.6 How MySQL optimizes LEFT JOIN

A LEFT JOIN B is in **MySQL** implemented as follows

- The table B is set to be dependent on table A.
- The table A is set to be dependent on all tables (except B) that are used in the LEFT JOIN condition.
- All LEFT JOIN conditions are moved to the WHERE clause.
- All standard join optimizations are done, with the exception that a table is always read after all tables it is dependent on. If there is a circular dependence then **MySQL** will issue an error.
- All standard WHERE optimzations are done.
- If there is a row in A that matches the WHERE clause, but there wasn't any row in B that matched the LEFT JOIN condition, then an extra B row is generated with all columns set to NULL.
- If you use LEFT JOIN to find rows that doesn't exist in some table and you have the following test: column_name IS NULL in the WHERE part, where column_name is a column that is declared as NOT NULL, then MySQL will stop searching after more rows (for a particular key combination) after it has found one row that matches the LEFT JOIN condition.

10.7 How MySQL optimizes LIMIT

In some cases **MySQL** will handle the query differently when you are using LIMIT # and not using HAVING:

- If you are selecting only a few rows with LIMIT, **MySQL** will use indexes in some cases when it normally would prefer to do a full table scan.
- If you use LIMIT # with ORDER BY, **MySQL** will end the sorting as soon as it has found the first # lines instead of sorting the whole table.
- When combinating LIMIT # with DISTINCT, **MySQL** will stop as soon as it finds # unique rows.
- As soon as MySQL has sent the first # rows to the client, it will abort the query.

10.8 How MySQL opens and closes tables

The cache of open tables can grow to a maximum of table_cache (default 64; this can be changed with with the -O table_cache=# option to mysqld). A table is never closed, except when the cache is full and another thread tries to open a table or if you use mysqladmin refresh or mysqladmin flush-tables.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables that are not currently in use are released, in least-recently-used order.
- If the cache is full and no tables can be released, but a new table needs to be opened, the cache is temporarily extended as necessary.

- If the cache is in a temporarily-extended state and a table goes from in-use to not-in-use state, it is closed and released from the cache.

A table is opened for each concurrent access. This means that if you have two threads accessing the same table or access the table twice in the same query (with `AS`) the table needs to be opened twice. The first open of any table takes two file descriptors; each additional use of the table takes only one file descriptor. The extra descriptor for the first open is used for the index file; this descriptor is shared among all threads.

10.8.1 Drawbacks of creating large numbers of tables in a database

If you have many files in a directory, open, close and create operations will be slow. If you execute `SELECT` statements on many different tables, there will be a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by making the table cache larger.

10.9 Why so many open tables?

When you run `mysqladmin status`, you'll see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082 Reloads: 1 Open tables: 12
```

This can be somewhat perplexing if you only have 6 tables.

MySQL is multithreaded, so it may have many queries on the same table simultaneously. To minimize the problem with two threads having different states on the same file, the table is opened independently by each concurrent thread. This takes some memory and one extra file descriptor for the data file. The index file descriptor is shared between all threads.

10.10 Using symbolic links for databases and tables

You can move tables and databases from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space.

If **MySQL** notices that a table is a symbolically-linked, it will resolve the symlink and use the table it points to instead. This works on all systems that support the `realpath()` call (at least Linux and Solaris support `realpath()`)! On systems that don't support `realpath()`, you should not access the table through the real path and through the symlink at the same time! If you do, the table will be inconsistent after any update.

MySQL doesn't support linking of databases by default. Things will work fine as long as you don't make a symbolic link between databases. Suppose you have a database `db1` under the **MySQL** data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

Now, for any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one thread updates `db1.tbl_a` and another thread updates `db2.tbl_a`, there will be problems.

If you really need this, you must change the following code in '`mysys/mf_format.c`':

```
if (!lstat(to,&stat_buff))  /* Check if it's a symbolic link */
    if (S_ISLNK(stat_buff.st_mode) && realpath(to,buff))
```

Change the code to this:

```
if (realpath(to,buff))
```

10.11 How MySQL locks tables

All locking in **MySQL** is deadlock-free. This is managed by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order.

The locking method **MySQL** uses for `WRITE` locks works as follows:

- If there are no locks on the table, put a write lock on it.
- Otherwise, put the lock request in the write lock queue.

The locking method **MySQL** uses for `READ` locks works as follows:

- If there are no write locks on the table, put a read lock on it.
- Otherwise, put the lock request in the read lock queue.

When a lock is released, the lock is made available to the threads in the write lock queue, then to the threads in the read lock queue.

This means that if you have many updates on a table, `SELECT` statements will wait until there are no more updates.

To work around this for the case where you want to do many `INSERT` and `SELECT` operations on a table, you can insert rows in a temporary table and update the real table with the records from the temporary table once in a while.

This can be done with the following code:

```
mysql> LOCK TABLES real_table WRITE, insert_table WRITE;
mysql> insert into real_table select * from insert_table;
mysql> delete from insert_table;
mysql> UNLOCK TABLES;
```

You can use the `LOW_PRIORITY` or `HIGH_PRIORITY` options with `INSERT` if you want to prioritize retrieval in some specific cases. See Section 7.13 [INSERT], page 179.

You could also change the locking code in '`mysys/thr_lock.c`' to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

10.12 How to arrange a table to be as fast/small as possible

You can get better performance on a table and minimize storage space using the techniques listed below:

- Declare columns to be `NOT NULL` if possible. It makes everything faster and you save one bit per column.
- Take advantage of the fact that all columns have default values. Insert values explicitly only when the value to be inserted differs from the default. You don't have to insert a value into the first `TIMESTAMP` column or into an `AUTO_INCREMENT` column in an `INSERT` statement. See Section 20.4.29 [mysql_insert_id()], page 348.

- Use the smaller integer types if possible to get smaller tables. For example, `MEDIUMINT` is often better than `INT`.
- If you don't have any variable-length columns (`VARCHAR`, `TEXT` or `BLOB` columns), a fixed-size record format is used. This is much faster but unfortunately may waste some space. See Section 10.17 [Row format], page 260.
- To help **MySQL** optimize queries better, run `isamchk --analyze` on a table after it has been loaded with relevant data. This updates a value for each index that indicates the average number of rows that have the same value. (For unique indexes, this is always 1, of course.)
- To sort an index and data according to an index, use `isamchk --sort-index --sort-records=1` (if you want to sort on index 1). If you have a unique index from which you want to read all records in order according to that index, this is a good way to make that faster. Note however that this sorting isn't written optimally and will take a long time for a large table!
- For `INSERT` statements, use multiple value lists if possible. This is much faster than using separate `INSERT` statements.
- When loading a table with data, use `LOAD DATA INFILE`. This is usually 20 times faster than using a lot of `INSERT` statements. See Section 7.15 [LOAD DATA], page 183.

  You can even get more speed when loading data into a table with many indexes using the following procedure:

  1. Create the table in `mysql` or Perl with `CREATE TABLE`.
  2. Execute `mysqladmin flush-tables`.
  3. Use `isamchk --keys-used=0 -rq /path/to/db/tbl_name`. This will remove all usage of all indexes from the table.
  4. Insert data into the table with `LOAD DATA INFILE`.
  5. If you have `pack_isam` and want to compress the table, run `pack_isam` on it.
  6. Recreate the indexes with `isamchk -r -q /path/to/db/tbl_name`.
  7. Execute `mysqladmin flush-tables`.
- To get some more speed for both `LOAD DATA INFILE` and `INSERT`, enlarge the key buffer. This can be done with the `-O key_buffer=#` option to `mysqld` or `safe_mysqld`. For example, 16M should be a good value if you have much RAM. :)
- When dumping data as text files for use by other programs, use `SELECT ... INTO OUTFILE`. See Section 7.15 [LOAD DATA], page 183.
- When doing many successive inserts or updates, you can get more speed by locking your tables using `LOCK TABLES`. `LOAD DATA INFILE` and `SELECT ...INTO OUTFILE` are atomic, so you don't have to use `LOCK TABLES` when using them. See Section 7.23 [LOCK TABLES], page 198.

To check how fragmented your tables are, run `isamchk -evi` on the '`.ISM`' file. See Chapter 13 [Maintenance], page 280.

10.13 Table locking issues

The table locking code in **MySQL** is deadlock free.

MySQL uses table locking (instead of row locking or column locking) to achieve a very high lock speed. For large tables, table locking is MUCH better than row locking, but there are of course some pitfalls.

Table locking enables many threads to read from a table at the same time, but if a thread wants to write to a table, it must first get exclusive access. During the update all others threads that want to access this particular table will wait until the update is ready.

As updates of databases normally are considered to be more important than `SELECT`, all statements that update a table have higher priority than statements that retrieve information from a table. This should ensure that updates are not 'starved' because one issues a lot of heavy queries against a specific table.

One main problem with this is the following:

- A client issues a `SELECT` that takes a long time to run.
- Another client then issues an `INSERT` on a used table; This client will wait until the `SELECT` is finished..
- Another client issues another `SELECT` statement on the same table; As `INSERT` has higher priority than `SELECT`, this `SELECT` will wait for the `INSERT` to finish. It will also wait for the first `SELECT` to finish!

Some possible solutions to this problem are:

- Try to get the `SELECT` statements to run faster; You may have to create some summary tables to do this.
- Start `mysqld` with `--low-priority-inserts`. This will give all statements that update a table lower priority than a `SELECT` statement. In this case the last `SELECT` statement in the previous scenario would execute before the `INSERT` statement.
- You can give a specific `INSERT`,`UPDATE` or `DELETE` statement lower priority with the `LOW_PRIORITY` attribute.
- You can specify that all updates from a specific thread should be done with low priority by using the SQL command: `SET SQL_LOW_PRIORITY_UPDATES=1`. See Section 7.24 [SET OPTION], page 199.
- You can specify that a specific `SELECT` is very important with the `HIGH_PRIORITY` attribute. See Section 7.11 [SELECT], page 176.
- If you mainly mix `INSERT` and `SELECT` statements, the `DELAYED` attribute to `INSERT` will probably solve your problems. See Section 7.13 [INSERT], page 179.
- If you have problems with `SELECT` and `DELETE`, the `LIMIT` option to `DELETE` may help. See Section 7.10 [DELETE], page 175.

10.14 Factors affecting the speed of `INSERT` statements

The time to insert a record consists of:

- Connect: (3)

- Sending query to server: (2)
- Parsing query: (2)
- Inserting record: (1 x size of record)
- Inserting indexes: (1 x indexes)
- Close: (1)

Where (number) is proportional time. This does not take into consideration the initial overhead to open tables (which is done once for each concurrently-running query).

The size of the table slows down the insertion of indexes by N log N (B-trees).

You can speed up insertions by locking your table and/or using multiple value lists with `INSERT` statements. Using multiple value lists can be up to 5 times faster than using separate inserts.

```
mysql> LOCK TABLES a WRITE;
mysql> INSERT INTO a VALUES (1,23),(2,34),(4,33);
mysql> INSERT INTO a VALUES (8,26),(6,29);
mysql> UNLOCK TABLES;
```

The main speed difference is that the index buffer is flushed to disk only once, after all `INSERT` statements have completed. Normally there would be as many index buffer flushes as there are different `INSERT` statements. Locking is not needed if you can insert all rows with a single statement.

Locking will also lower the total time of multi-connection tests, but the maximum wait time for some threads will go up (because they wait for locks). For example:

```
thread 1 does 1000 inserts
thread 2, 3, and 4 does 1 insert
thread 5 does 1000 inserts
```

If you don't use locking, 2, 3 and 4 will finish before 1 and 5. If you use locking, 2, 3 and 4 probably will not finish before 1 or 5, but the total time should be about 40% faster.

As `INSERT`, `UPDATE` and `DELETE` operations are very fast in **MySQL**, you will obtain better overall performance by adding locks around everything that does more than about 5 inserts or updates in a row. If you do very many inserts in a row, you could do a `LOCK TABLES` followed by a `UNLOCK TABLES` once in a while (about each 1000 rows) to allow other threads access to the table. This would still result in a nice performance gain.

Of course, `LOAD DATA INFILE` is much faster still.

If you are inserting a lot of rows from different clients, you can get higher speed by using the `INSERT DELAYED` statement. See Section 7.13 [INSERT], page 179.

10.15 Factors affecting the speed of `DELETE` statements

The time to delete a record is exactly proportional to the number of indexes. To delete records more quickly, you can increase the size of the index cache. The default index cache is 1M; to get faster deletes, it should be increased by several factors (try 16M if you have enough memory).

10.16 How do I get MySQL to run at full speed?

Start by benchmarking your problem! You can take any program from the **MySQL** benchmark suite (normally found in the 'sql-bench' directory) and modify it for your needs. By doing this, you can try different solutions to your problem and test which is really the fastest solution for you.

- Start `mysqld` with the correct options. More memory gives more speed if you have it. See Section 10.1 [Server parameters], page 244.
- Create indexes to make your `SELECT` statements faster. See Section 10.4 [**MySQL** indexes], page 249.
- Optimize your column types to be as efficient as possible. For example, declere columns to be `NOT NULL` if possible. See Section 10.12 [Table efficiency], page 255.
- **MySQL** has two different levels of locks: Internal locks and external locks. The internal locks ensures that all update/retrieve operations are atomic (run without conflicts from other clients). The external locks allows multiple **MySQL** servers to run on the same data and allows you to use `isamchk` to check tables without taking down `mysqld`.

  The `--skip-locking` option disables external locking (file locking) between SQL requests. This gives greater speed but has the following consequences:

  – You **MUST** flush all tables with `mysqladmin flush-tables` before you try to check or repair tables with `isamchk`. (`isamchk -d tbl_name` is always allowed, since that simply displays table information.)

  – You can't run two **MySQL** servers on the same data files, if both are going to update the same tables.

  The `--skip-locking` option is on by default when compiling with MIT-pthreads, because `flock()` isn't fully supported by MIT-pthreads on all platforms.

  The only case when you can't use `--skip-locking` is if you run multiple **MySQL** SERVERS (not clients) on the same data (or run `isamchk` on the table without first flushing the `mysqld` server tables first).

  You can still use `LOCK TABLES` / `UNLOCK TABLES` even if you are using `--skip-locking`
- If updates are a problem, you can delay updates and then do many updates in a row later. Doing many updates in a row is much quicker than doing one at a time.
- On FreeBSD systems, if the problem is with MIT-pthreads, upgrading to FreeBSD 3.0 (or higher) should help. This makes it possible to use Unix sockets (with FreeBSD, this is quicker than connecting using TCP/IP with MIT-pthreads) and the threads package is much more integrated.
- `GRANT` checking on the table or column level will decrease performance.

If your problem is with some explicit **MySQL** function, you can always time this in the **MySQL** client:

```
mysql> select benchmark(1000000,1+1);
+------------------------+
| benchmark(1000000,1+1) |
+------------------------+
|                      0 |
```

```
+------------------------+
1 row in set (0.32 sec)
```

The above shows that **MySQL** can execute 1,000,000 `+` expressions in 0.32 seconds on a simple `PentiumII 400MHz`.

All **MySQL** functions should be very optimized, but there may be some exceptions and the `benchmark(loop_count,expression)` is a great tool to find if this is a problem with your query.

10.17 What are the different row formats? Or, when should `VARCHAR/CHAR` be used?

MySQL dosen't have true SQL `VARCHAR` types.

Instead, **MySQL** has three different ways to store records and uses these to emulate `VARCHAR`.

If a table doesn't have any `VARCHAR`, `BLOB` or `TEXT` columns, a fixed row size is used. Otherwise a dynamic row size is used. `CHAR` and `VARCHAR` columns are treated identically from the application's point of view; both have trailing spaces removed when the columns are retrieved.

You can check the format used in a table with `isamchk -d` (`-d` means "describe the table").

MySQL has three different table formats: fixed-length, dynamic and compressed. These are compared below.

Fixed-length tables

- This is the default format. It's used when the table contains no `VARCHAR`, `BLOB` or `TEXT` columns.
- All `CHAR`, `NUMERIC` and `DECIMAL` columns are space-padded to the column width.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because records are located in fixed positions.
- Doesn't have to be reorganized (with `isamchk`) unless a huge number of records are deleted and you want to return free disk space to the operating system.
- Usually requires more disk space than dynamic tables.

Dynamic tables

- This format is used if the table contains any `VARCHAR`, `BLOB` or `TEXT` columns.
- All string columns are dynamic (except those with a length less than 4).
- Each record is preceded by a bitmap indicating which columns are empty (`''`) for string columns, or zero for numeric columns (this isn't the same as columns containing `NULL` values). If a string column has a length of zero after removal of trailing spaces, or a numeric column has a value of zero, it is marked in the bit map and not saved to disk. Non-empty strings are saved as a length byte plus the string contents.
- Usually takes much less disk space than fixed-length tables.
- Each record uses only as much space as is required. If a record becomes larger, it is split into as many pieces as required. This results in record fragmentation.

- If you update a row with information that extends the row length, the row will be fragmented. In this case, you may have to run `isamchk -r` from time to time to get better performance. Use `isamchk -ei tbl_name` for some statistics.
- Not as easy to reconstruct after a crash, because a record may be fragmented into many pieces and a link (fragment) may be missing.
- The expected row length for dynamic sized records is:

```
3
+ (number of columns + 7) / 8
+ (number of char columns)
+ packed size of numeric columns
+ length of strings
+ (number of NULL columns + 7) / 8
```

  There is a penalty of 6 bytes for each link. A dynamic record is linked whenever an update causes an enlargement of the record. Each new link will be at least 20 bytes, so the next enlargement will probably go in the same link. If not, there will be another link. You may check how many links there are with `isamchk -ed`. All links may be removed with `isamchk -r`.

Compressed tables

- A read-only table made with the `pack_isam` utility. All customers with extended **MySQL** email support are entitled to a copy of `pack_isam` for their internal usage.
- The uncompress code exists in all **MySQL** distributions so that even customers who don't have `pack_isam` can read tables that were compressed with `pack_isam` (as long as the table was compressed on the same platform).
- Takes very little disk space. Minimises disk usage.
- Each record is compressed separately (very little access overhead). The header for a record is fixed (1-3 bytes) depending on the biggest record in the table. Each column is compressed differently. Some of the compression types are:
  - There is usually a different Huffman table for each column.
  - Suffix space compression.
  - Prefix space compression.
  - Numbers with value `0` are stored using 1 bit.
  - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (8 bytes) may be stored as a `TINYINT` column (1 byte) if all values are in the range `0` to `255`.
  - If a column has only a small set of possible values, the column type is converted to `ENUM`.
  - A column may use a combination of the above compressions.
- Can handle fixed or dynamic length records, but not `BLOB` or `TEXT` columns.
- Can be uncompressed with `isamchk`.

MySQL can support different index types, but the normal type is NISAM. This is a B-tree index and you can roughly calculate the size for the index file as `(key_length+4)*0.67`, summed over all keys. (This is for the worst case when all keys are inserted in sorted order.)

String indexes are space compressed. If the first index part is a string, it will also be prefix compressed. Space compression makes the index file smaller if the string column has a lot of trailing space or is a `VARCHAR` column that is not always used to the full length. Prefix compression helps if there are many strings with an identical prefix.

10.18 MySQL table types.

Table types are introduced in **MySQL** 3.23!

When you create a new table, you can tell **MySQL** which table type it should use for the table. **MySQL** will always create a `.frm` file to hold the table and column definitions. Depending on the table type the index and data will be stored in other files.

You can convert tables between different types with the `ALTER TABLE` statement. See Section 7.7 [ALTER TABLE], page 172.

- `ISAM`

  This is the original **MySQL** table type. This uses a `B-tree` index. The index is stored in a file with the `.ISM` extension and the data is stored in file with the `.ISD` extension. You can check/repair `ISAM` tables with the `isamchk` utility. See Section 13.4 [Crash recovery], page 289. `ISAM` tables are not binary portable across OS/Platforms.

  `ISAM` has the following features/properties:

  - Compressed and fixed length keys
  - Fixed and dynamic record length
  - 16 keys with 16 key parts / key
  - Max key length 256 (default)
  - Data is stored in machine format; Fast but is machine/OS dependent.

- `MyISAM` `MyISAM` is the default table type in **MySQL** 3.23. It's based on the `ISAM` code and has a lot of useful extensions.

  The index is stored in a file with the `.MYI` extension and the data is stored in file with the `.MYD` extension. You can check/repair `ISAM` tables with the `myisamchk` utility. See Section 13.4 [Crash recovery], page 289

  The following is new in `MyISAM`:

  - Supports for big files (63 bit) on OSes that support big files.
  - All data are stored with low byte first. This makes the data machine and OS independent.
  - All number keys are stored with high byte first to give better compression.
  - Internal handling of one `AUTO_INCREMENT` column. `MyISAM` will automatically update this on `INSERT/UPDATE`. The `AUTO_INCREMENT` value can be reset with `myisamchk`. This will make `AUTO_INCREMENT` columns faster and old numbers will not be reused.
  - Maximum key length is now 500 by default. In cases of keys longer than 250, a bigger key block size than the default of 1024 bytes is used for this key.
  - Maximum number of keys/table enlarged to 32 as default. This can be enlarged to 64 without having to recompile `myisamchk`.

- There is a flag in the `MyISAM` file that indicates whether or not the table was closed correctly. This will soon be used for automatic repair in the **MySQL** server.
- `myisamchk` will now mark tables as checked. `myisamchk --fast` will only check those tables that don't have this mark.
- `myisamchk -a` stores statistics for key parts (and not only for whole keys as in `NISAM`).
- Dynamic size rows will now be much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.
- `myisampack` (called `pack_isam` in `NISAM`) can pack `BLOB` and `VARCHAR`columns.

`MyISAM` also supports the following things, which **MySQL** will be able to use in the near future.

- `BLOB` and `TEXT` columns can be indexed.
- Support for a true `VARCHAR` type; A `VARCHAR` column starts with a length stored in 2 bytes.
- Tables with `VARCHAR` may have fixed or dynamic record length.
- `VARCHAR` and `CHAR` may be up to 64K. All key segments have their own language definition. This will enable **MySQL** to have different language definitions per column.
- `NULL` values are allowed in indexed columns. This takes 0-1 bytes/key.
- A hashed computed index can be used for `UNIQUE`; This will allow you to have `UNIQUE` on any combination of columns in a table. (You can't search on a `UNIQUE` computed index, however.)

- `HEAP` `HEAP` tables use a hashed index and are stored in memory. This makes them very fast, but if **MySQL** crashes you will lose all data stored in them. `HEAP` is very usable as temporary tables!

```
CREATE TABLE test TYPE=HEAP SELECT ip,SUM(downloads) as down FROM log_table GROU
SELECT COUNT(ip),AVG(down) from test;
drop table test;
```

Here are some things you should consider when you use `HEAP` tables:

- You should always use specify `MAX_ROWS` in the `CREATE` statement to ensure that you accidently do not use all memory.
- Indexes will only be used with `=` and `<=>` (but are VERY fast).
- `HEAP` tables uses a fixed record length format.
- `HEAP` doesn't support `BLOB/TEXT` columns.
- `HEAP` doesn't support `AUTO_INCREMENT` columns.
- `HEAP` doesn't support an index on a `NULL` column.
- You can have non-unique keys in a `HEAP` table (not that normal with hashed tables).
- `HEAP` tables are shared between all clients (just like any other table).
- Data for `HEAP` tables are allocated in small blocks. The tables are 100% dynamic (on inserting). No overflow areas and no extra key space is needed. Deleted rows are put in a linked list and will be reused when you insert new data into the table.

- To free memory, you should execute `DELETE FROM heap_table` or `DROP TABLE heap_table`.
- To ensure that you accidentally don't do anything stupid, you can't create `HEAP` tables bigger than `max_heap_table_size`.

11 The MySQL benchmark suite

This should contain a technical description of the **MySQL** benchmark suite (and `crash-me`) but that description is not written yet. Currently, you should look at the code and results in the '`bench`' directory in the distribution (and of course on the web page at http://www.mysql.com/crash-me-choose.htmy).

It is meant to be a benchmark that will tell any user what things a given SQL implementation performs well or poorly at.

`crash-me` tries to determine what features a database supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What column types are supported
- How many indexes are supported
- What functions are supported
- How big a query can be
- How big a `VARCHAR` column can be

12 MySQL Utilites

12.1 Overview of the different MySQL programs

All **MySQL** clients that communicate with the server using the `mysqlclient` library use the following environment variables:

Name	Description
`MYSQL_UNIX_PORT`	The default socket; used for connections to `localhost`
`MYSQL_TCP_PORT`	The default TCP/IP port
`MYSQL_PWD`	The default password
`MYSQL_DEBUG`	Debug-trace options when debugging
`TMPDIR`	The directory where temporary tables/files are created

Use of `MYSQL_PWD` is insecure. See Section 6.3 [Connecting], page 94.

The '`mysql`' client uses the file named in the `MYSQL_HISTFILE` environment variable to save the command line history. The default value for the history file is '`$HOME/.mysql_history`', where `$HOME` is the value of the `HOME` environment variable.

All **MySQL** programs take many different options. However, every **MySQL** program provides a `--help` option that you can use to get a full description of the program's different options. For example, try `mysql --help`.

You can override default options for all standard client programs with an option file. Section 4.15.4 [Option files], page 79.

The list below briefly describes the **MySQL** programs:

`isamchk` Utility to describe, check, optimize and repair **MySQL** tables. Because `isamchk` has many functions, it is described in its own chapter. See Chapter 13 [Maintenance], page 280.

`make_binary_release`
Makes a binary release of a compiled **MySQL**. This could be sent by FTP to '`/pub/mysql/Incoming`' on `ftp.tcx.se` for the convenience of other **MySQL** users.

`msql2mysql`
A shell script that converts `mSQL` programs to **MySQL**. It doesn't handle all cases, but it gives a good start when converting.

`mysql` `mysql` is a simple SQL shell (with GNU `readline` capabilities). It supports interactive and non-interactive use. When used interactively, query results are presented in an ASCII-table format. When used non-interactively (e.g., as a filter), the result is presented in tab-separated format. (The output format can be changed using command-line options.) You can run scripts simply like this:

```
shell> mysql database < script.sql > output.tab
```

If you have problems due to insufficient memory in the client, use the `--quick` option! This forces `mysql` to use `mysql_use_result()` rather than `mysql_store_result()` to retrieve the result set.

`mysqlaccess`
: A script that checks the access privileges for a host, user and database combination.

`mysqladmin`
: Utility for performing administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk and reopening log files. `mysqladmin` can also be used to retrieve version, process and status information from the server. See Section 12.2 [mysqladmin], page 267.

`mysqlbug`
: The **MySQL** bug report script. This script should always be used when filing a bug report to the **MySQL** list.

`mysqld`
: The SQL daemon. This should always be running.

`mysqldump`
: Dumps a **MySQL** database into a file as SQL statements or as tab-separated text files. Enhanced freeware originally by Igor Romanenko. See Section 12.3 [mysqldump], page 268.

`mysqlimport`
: Imports text files into their respective tables using `LOAD DATA INFILE`. See Section 12.4 [mysqlimport], page 271.

`mysqlshow`
: Displays information about databases, tables, columns and indexes.

`mysql_install_db`
: Creates the **MySQL** grant tables with default privileges. This is usually executed only once, when first installing **MySQL** on a system.

`replace`
: A utility program that is used by `msql2mysql`, but that has more general applicability as well. `replace` changes strings in place in files or on the standard input. Uses a finite state machine to match longer strings first. Can be used to swap strings. For example, this command swaps `a` and `b` in the given files:

```
shell> replace a b b a -- file1 file2 ...
```

`safe_mysqld`
: A script that starts the `mysqld` daemon with some safety features, such as restarting the server when an error occurs and logging runtime information to a log file.

12.2 Administering a MySQL server

Utility for performing administrative operations. The syntax is:

```
shell> mysqladmin [OPTIONS] command [command-option] command ...
```

You can get a list of the options your version of `mysqladmin` supports by executing `mysqladmin --help`.

The current `mysqladmin` supports the following commands:

create databasename Create a new database.

drop databasename	Delete a database and all its tables.
extended-status	Gives an extended status message from the server.
flush-hosts	Flush all cached hosts.
flush-logs	Flush all logs.
flush-tables	Flush all tables.
flush-privileges	Reload grant tables (same as reload)
kill id,id,...	Kill mysql threads.
password	new-password Change old password to new-password
ping	Check if mysqld is alive
processlist	Show list of active threads in server
reload	Reload grant tables
refresh	Flush all tables and close and open logfiles
shutdown	Take server down
status	Gives a short status message from the server
variables	Prints variables available
version	Get version info from server

All commands can be shortened to their unique prefix. For example:

```
shell> mysqladmin proc stat
+----+-------+-----------+----+-------------+------+-------+------+
| Id | User  | Host      | db | Command     | Time | State | Info |
+----+-------+-----------+----+-------------+------+-------+------+
| 6  | monty | localhost |    | Processlist | 0    |       |      |
+----+-------+-----------+----+-------------+------+-------+------+
Uptime: 10077  Threads: 1  Questions: 9  Slow queries: 0  Opens: 6  Flush tables: 1
```

The `mysqladmin status` command result has the following columns:

Uptime	Number of seconds the **MySQL** server have been up
Threads	Number of active threads (clients)
Questions	Number of questions from clients since `mysqld` was started
Slow queries	Queries that has taken more than `long_query_time` seconds
Opens	How many tables `mysqld` has opened.
Flush tables	Number of `flush ...`, `refresh` and `reload` commands.
Open tables	Number of tables that are open now
Memory in use	Memory allocated directly by the mysqld code (only available when **MySQL** is compiled with –with-debug)
Max memory used	Maximum memory allocated directly by the mysqld code (only available when **MySQL** is compiled with –with-debug)

12.3 Dumping the structure and data from MySQL databases and tables

Utility to dump a database or a collection of database for backup or for transferring the data to another SQL server. The dump will contain SQL statements to create the table and/or populate the table.

```
shell> mysqldump [OPTIONS] database [tables]
```

If you don't give any tables, the whole database will be dumped.

You can get a list of the options your version of `mysqldump` supports by executing `mysqldump --help`.

Note that if you run `mysqldump` without `--quick` or `--opt`, `mysqldump` will load the whole result set into memory before dumping the result. This will probably be a problem if you are dumping a big database.

`mysqldump` supports the following options:

`--add-locks`
: Add `LOCK TABLES` before and `UNLOCK TABLE` after each table dump. (To get faster inserts into **MySQL**).

`--add-drop-table`
: Add a `drop table` before each create statement.

`--allow-keywords`
: Allow creation of column names that are keywords. This works by prefixing each column name with the table name.

`-c, --complete-insert`
: Use complete insert statements (with column names).

`-C, --compress`
: Compress all information between the client and the server if both support compression.

`--delayed`
: Insert rows with the `INSERT DELAYED` command.

`-e, --extended-insert`
: Use the new multiline `INSERT` syntax. (Gives more compact and faster inserts statements)

`-#, --debug[=option_string]`
: Trace usage of the program (for debugging).

`--help` Display a help message and exit.

`--fields-terminated-by=...`
`--fields-enclosed-by=...`
`--fields-optionally-enclosed-by=...`
`--fields-escaped-by=...`
`--fields-terminated-by=...`
: These options are used with the `-T` option and have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See Section 7.15 [LOAD DATA], page 183.

`-F, --flush-logs`
: Flush logs file in the **MySQL** server before starting the dump.

`-f, --force,`
: Continue even if we get an SQL error during a table dump.

`-h, --host=..`
: Dump data from the **MySQL** server on the named host. The default host is `localhost`.

`-l, --lock-tables.`
: Lock all tables for starting the dump.

`-t, --no-create-info`
: Don't write table creation info (The `CREATE TABLE` statment)

`-d, --no-data`
: Don't write any row information for the table. This is very useful if you just want to get a dump of the structure for a table!

`--opt`
: Same as `--quick --add-drop-table --add-locks --extended-insert --use-locks`. Should give you the fastest possible dump for reading into a **MySQL** server.

`-pyour_pass, --password[=your_pass]`
: The password to use when connecting to the server. If you specify no '`=your_pass`' part, `mysqldump` solicits the password from the terminal.

`-P port_num, --port=port_num`
: The TCP/IP port number to use for connecting to a host. (This is used for connections to hosts other than `localhost`, for which Unix sockets are used.)

`-q, --quick`
: Don't buffer query, dump directly to stdout; Uses `mysql_use_result()` to do this.

`-S /path/to/socket, --socket=/path/to/socket`
: The socket file to use when connecting to `localhost` (which is the default host).

`-T, --tab=path-to-some-directory`
: Creates a `table_name.sql` file, that conntains the SQL CREATE commands, and a `table_name.txt` file, that contains the data, for each give table. **NOTE**: This only works if `mysqldump` is run on the same machine as the `mysqld` daemon. The format of the `.txt` file is made according to the `--fields-xxx` and `--lines--xxx` options.

`-u user_name, --user=user_name`
: The **MySQL** user name to use when connecting to the server. The default value is your Unix login name.

`-O var=option, --set-variable var=option`
: Set the value of a variable. The possible variables are listed below.

`-v, --verbose`
: Verbose mode. Print out more information what the program does.

`-V, --version`
: Print version information and exit.

`-w, --where='where-condition'`
: Dump only selected records; Note that QUOTES are mandatory!

```
"--where=user='jimf'" "-wuserid>1" "-wuserid<1"
```

The most normal use of `mysqldump` is probably for making a backup of whole database:

```
mysqldump --opt database > backup-file.sql
```

But it's also very useful to populate another **MySQL** server with information from a databas:

```
mysqldump --opt database | mysql ---host=remote-host -C database
```

12.4 Importing data from text files

`mysqlimport` provides a command line interface to the `LOAD DATA INFILE` SQL statement. Most options to `mysqlimport` correspond directly to the same options to `LOAD DATA INFILE`. See Section 7.15 [LOAD DATA], page 183.

`mysqlimport` is invoked like this:

```
shell> mysqlimport [options] filename ...
```

For each text file named on the command line, `mysqlimport` strips any extension from the filename and uses the result to determine which table to import the file's contents into. For example, files named '`patient.txt`', '`patient.text`' and '`patient`' would all be imported into a table named `patient`.

`mysqlimport` supports the following options:

`-C, --compress`
: Compress all information between the client and the server if both support compression.

`-#, --debug[=option_string]`
: Trace usage of the program (for debugging).

`-d, --delete`
: Empty the table before importing the text file.

`--fields-terminated-by=...`
`--fields-enclosed-by=...`
`--fields-optionally-enclosed-by=...`
`--fields-escaped-by=...`
`--fields-terminated-by=...`
: These options have the same meaning as the corresponding clauses for `LOAD DATA INFILE`. See Section 7.15 [LOAD DATA], page 183.

`-f, --force`
: Ignore errors. For example, if a table for a text file doesn't exist, continue processing any remaining files. Without `--force`, `mysqlimport` exits if a table doesn't exist.

`--help` Display a help message and exit.

`-h host_name, --host=host_name`
: Import data to the **MySQL** server on the named host. The default host is `localhost`.

`-i, --ignore`
: See the description for the `--replace` option.

`-l, --lock-tables`
: Lock **ALL** tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

`-L, --local`
: Read input files from the client. By default, text files are assumed to be on the server if you connect to `localhost` (which is the default host).

`-pyour_pass, --password[=your_pass]`
: The password to use when connecting to the server. If you specify no '`=your_pass`' part, `mysqlimport` solicits the password from the terminal.

`-P port_num, --port=port_num`
: The TCP/IP port number to use for connecting to a host. (This is used for connections to hosts other than `localhost`, for which Unix sockets are used.)

`-r, --replace`
: The `--replace` and `--ignore` options control handling of input records that duplicate existing records on unique key values. If you specify `--replace`, new rows replace existing rows that have the same unique key value. If you specify `--ignore`, input rows that duplicate an existing row on a unique key value are skipped. If you don't specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

`-s, --silent`
: Silent mode. Write output only when errors occur.

`-S /path/to/socket, --socket=/path/to/socket`
: The socket file to use when connecting to `localhost` (which is the default host).

`-u user_name, --user=user_name`
: The **MySQL** user name to use when connecting to the server. The default value is your Unix login name.

`-v, --verbose`
: Verbose mode. Print out more information what the program does.

`-V, --version`
: Print version information and exit.

12.5 The MySQL compressed read-only table generator

`pack_isam` is an extra utility that you get when you order more than 10 licenses or extended support. Since `pack_isam` is distributed only in binary form, `pack_isam` is available only on some platforms.

Of course, all future updates to `pack_isam` are included in the price. `pack_isam` may at some time be included as standard when we get some kind of turnover for **MySQL**.

`pack_isam` works by compressing each column in the table separately. The information needed to decompress columns is read into memory when the table is opened. This results in much better performance when accessing individual records, since you only have to uncompress exactly one record, not a much larger disk block like when using Stacker on MS-DOS. Usually, `pack_isam` packs the data file 40%-70%.

MySQL uses memory mapping (`mmap()`) on compressed tables and falls back to normal read/write file usage if `mmap()` doesn't work.

There are currently two limitations with `pack_isam`:

- After packing, the table is read only.
- It can't pack `BLOB` columns, yet.

Fixing these limitations is on our TODO list but with low priority.

`pack_isam` is invoked like this:

```
shell> pack_isam [options] filename ...
```

Each filename should be the name of an index ('`.ISM`') file. If you are not in the database directory, you should specify the pathname to the file. It is permissible to omit the '`.ISM`' extension.

`pack_isam` supports the following options:

`-b, --backup`
: Make a backup of the table as `tbl_name.OLD`.

`-#, --debug=debug_options`
: Output debug log. The `debug_options` string often is `'d:t:o,filename'`.

`-f, --force`
: Force packing of the table even if it becomes bigger or if the temporary file exists. (`pack_isam` creates a temporary file named '`tbl_name.TMD`' while it compresses the table. If you kill `pack_isam`, the '`.TMD`' file may not be deleted. Normally, `pack_isam` exits with an error if it finds that '`tbl_name.TMD`' exists. With `--force`, `pack_isam` packs the table anyway.

`-?, --help`
: Display a help message and exit.

`-j big_tbl_name, --join=big_tbl_name`
: Join all tables named on the command line into a single table `big_tbl_name`. All tables that are to be combined MUST be identical (same column names and types, same indexes, etc.)

`-p #, --packlength=#`
: Specify the record length storage size, in bytes. The value should be 1, 2 or 3. (`pack_isam` stores all rows with length pointers of 1, 2 or 3 bytes. In most normal cases, `pack_isam` can determine the right length value before it begins packing the file, but it may notice during the packing process that it could have used a shorter length. In this case, `pack_isam` will print a note that the next time you pack the same file, you could use a shorter record length.)

`-s, --silent`
: Silent mode. Write output only when errors occur.

`-t, --test`
: Don't pack table, only test packing it.

`-T dir_name, --tmp_dir=dir_name`
: Use the named directory as the location in which to write the temporary table.

`-v, --verbose`
: Verbose mode. Write info about progress and packing result.

`-V, --version`
: Display version information and exit.

`-w, --wait`
: Wait and retry if table is in use. If the `mysqld` server was invoked with the `--skip-locking` option, it is not a good idea to invoke `pack_isam` if the table might be updated during the packing process.

The sequence of commands shown below illustrates a typical table compression session:

```
shell> ls -l station.*
-rw-rw-r--   1 monty    my         994128 Apr 17 19:00 station.ISD
-rw-rw-r--   1 monty    my          53248 Apr 17 19:00 station.ISM
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> isamchk -dvv station

ISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-02-02  3:06:43
Data records:              1192  Deleted blocks:              0
Datafile: Parts:           1192  Deleted data:                0
Datafile pointer (bytes):     2  Keyfile pointer (bytes):     2
Max datafile length:   54657023  Max keyfile length:   33554431
Recordlength:               834
Record format: Fixed length

table description:
Key Start Len Index   Type                     Root  Blocksize    Rec/key
1   2     4   unique  unsigned long             1024       1024          1
2   32    30  multip. text                     10240       1024          1

Field Start Length Type
1     1     1
2     2     4
3     6     4
4     10    1
5     11    20
6     31    1
```

```
7       32      30
8       62      35
9       97      35
10      132     35
11      167     4
12      171     16
13      187     35
14      222     4
15      226     16
16      242     20
17      262     20
18      282     20
19      302     30
20      332     4
21      336     4
22      340     1
23      341     8
24      349     8
25      357     8
26      365     2
27      367     2
28      369     4
29      373     4
30      377     1
31      378     2
32      380     8
33      388     4
34      392     4
35      396     4
36      400     4
37      404     1
38      405     4
39      409     4
40      413     4
41      417     4
42      421     4
43      425     4
44      429     20
45      449     30
46      479     1
47      480     1
48      481     79
49      560     79
50      639     79
51      718     79
52      797     8
53      805     1
54      806     1
```

```
55    807   20
56    827   4
57    831   4

shell> pack_isam station.ISM
Compressing station.ISM: (1192 records)
- Calculating statistics

normal:     20  empty-space:      16  empty-zero:        12  empty-fill:  11
pre-space:   0  end-space:        12  table-lookups:      5  zero:         7
Original trees:  57  After join: 17
- Compressing file
87.14%

shell> ls -l station.*
-rw-rw-r--   1 monty    my         127874 Apr 17 19:00 station.ISD
-rw-rw-r--   1 monty    my          55296 Apr 17 19:04 station.ISM
-rw-rw-r--   1 monty    my           5767 Apr 17 19:00 station.frm

shell> isamchk -dvv station

ISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-04-17 19:04:26
Data records:              1192  Deleted blocks:              0
Datafile: Parts:           1192  Deleted data:                0
Datafilepointer (bytes):      3  Keyfile pointer (bytes):     1
Max datafile length:   16777215  Max keyfile length:     131071
Recordlength:               834
Record format: Compressed

table description:
Key Start Len Index   Type                        Root  Blocksize    Rec/key
1   2     4   unique  unsigned long              10240       1024          1
2   32    30  multip. text                       54272       1024          1

Field Start Length Type                          Huff tree  Bits
1     1     1      constant                            1     0
2     2     4      zerofill(1)                         2     9
3     6     4      no zeros, zerofill(1)               2     9
4     10    1                                          3     9
5     11    20     table-lookup                        4     0
6     31    1                                          3     9
7     32    30     no endspace, not_always             5     9
8     62    35     no endspace, not_always, no empty   6     9
9     97    35     no empty                            7     9
10    132   35     no endspace, not_always, no empty   6     9
```

```
11    167   4      zerofill(1)                          2     9
12    171   16     no endspace, not_always, no empty    5     9
13    187   35     no endspace, not_always, no empty    6     9
14    222   4      zerofill(1)                          2     9
15    226   16     no endspace, not_always, no empty    5     9
16    242   20     no endspace, not_always              8     9
17    262   20     no endspace, no empty                8     9
18    282   20     no endspace, no empty                5     9
19    302   30     no endspace, no empty                6     9
20    332   4      always zero                          2     9
21    336   4      always zero                          2     9
22    340   1                                           3     9
23    341   8      table-lookup                         9     0
24    349   8      table-lookup                        10     0
25    357   8      always zero                          2     9
26    365   2                                           2     9
27    367   2      no zeros, zerofill(1)                2     9
28    369   4      no zeros, zerofill(1)                2     9
29    373   4      table-lookup                        11     0
30    377   1                                           3     9
31    378   2      no zeros, zerofill(1)                2     9
32    380   8      no zeros                             2     9
33    388   4      always zero                          2     9
34    392   4      table-lookup                        12     0
35    396   4      no zeros, zerofill(1)               13     9
36    400   4      no zeros, zerofill(1)                2     9
37    404   1                                           2     9
38    405   4      no zeros                             2     9
39    409   4      always zero                          2     9
40    413   4      no zeros                             2     9
41    417   4      always zero                          2     9
42    421   4      no zeros                             2     9
43    425   4      always zero                          2     9
44    429   20     no empty                             3     9
45    449   30     no empty                             3     9
46    479   1                                          14     4
47    480   1                                          14     4
48    481   79     no endspace, no empty               15     9
49    560   79     no empty                             2     9
50    639   79     no empty                             2     9
51    718   79     no endspace                         16     9
52    797   8      no empty                             2     9
53    805   1                                          17     1
54    806   1                                           3     9
55    807   20     no empty                             3     9
56    827   4      no zeros, zerofill(2)                2     9
57    831   4      no zeros, zerofill(1)                2     9
```

The information printed by `pack_isam` is described below:

`normal` The number of columns for which no extra packing is used.

`empty-space`
: The number of columns containing values that are only spaces; these will occupy 1 bit.

`empty-zero`
: The number of columns containing values that are only binary 0's; these will occupy 1 bit.

`empty-fill`
: The number of integer columns that don't occupy the full byte range of their type; these are changed to a smaller type (for example, an `INTEGER` column may be changed to `MEDIUMINT`).

`pre-space`
: The number of decimal columns that are stored with leading space. In this case, each value will contain a count for the number of leading spaces.

`end-space`
: The number of columns that have a lot of trailing space. In this case, each value will contain a count for the number of trailing spaces.

`table-lookup`
: The column had only a small number of different values, and that were converted to an `ENUM` before Huffman compression.

`zero` The number of columns for which all values are zero.

`Original trees`
: The initial number of Huffman trees.

`After join`
: The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, `isamchk -dvv` prints additional information about each field:

`Type` The field type may contain the following descriptors:

`constant` All rows have the same value.

`no endspace`
: Don't store endspace.

`no endspace, not_always`
: Don't store endspace and don't do end space compression for all values.

`no endspace, no empty`
: Don't store endspace. Don't store empty values.

`table-lookup`
: The column was converted to an `ENUM`.

`zerofill(n)`
: The most significant `n` bytes in the value are always 0 and are not stored.

`no zeros`
: Don't store zeros.

`always zero`
: 0 values are stored in 1 bit.

`Huff tree`
: The Huffman tree associated with the field

`Bits`
: The number of bits used in the Huffman tree.

13 Maintaining a MySQL installation

13.1 Using `isamchk` for table maintenance and crash recovery

To check/repair ISAM tables (`.ISM` and `.ISD`) you should use the `isamchk` utility. To check/repair MyISAM tables (`.MYI` and `.MYD`) you should use the `myisamchk` utility. See Section 10.18 [Table types], page 262.

In the following text we will talk about `isamchk` but everything also applies to `myisamchk`.

You can use the `isamchk` utility to get information about your database tables, check and repair them or optimize them. The following sections describe how to invoke `isamchk` (including a description of its options), how to set up a table maintenance schedule, and how to use `isamchk` to perform its various functions.

If you run `mysqld` with `--skip-locking` (which is the default on some systems, like Linux), you can't reliably use `isamchk` to check a table when `mysqld` is using the same table. If you can be sure that no one is accessing the tables through `mysqld` while you run `isamchk`, you only have to do `mysqladmin flush-tables` before you start checking the tables. If you can't guarantee the above, then you must take down `mysqld` while you check the tables. If you run `isamchk` while `mysqld` is updating the tables, you may get a warning that a table is corrupt even if it isn't.

If you are not using `--skip-locking`, you can use `isamchk` to check tables at any time. While you do this, all clients that try to update the table will wait until `isamchk` is ready before continuing.

If you use `isamchk` to repair or optimize tables, you **MUST** always ensure that the `mysqld` server is not using the table (this also applies if you are using `--skip-locking`). If you don't take down `mysqld` you should at least do a `mysqladmin flush-tables` before you run `isamchk`.

You can in most cases also use the command `OPTIMIZE TABLES` to optimize and repair tables, but this is not as fast or reliable (in case of real fatal errors) as `isamchk`. On the other hand, `OPTIMIZE TABLE` is easier to use and you don't have to worry about flushing tables. See Section 7.8 [`OPTIMIZE TABLE`], page 175.

13.1.1 `isamchk` invocation syntax

`isamchk` is invoked like this:

```
shell> isamchk [options] tbl_name
```

The `options` specify what you want `isamchk` to do. They are described below. (You can also get a list of options by invoking `isamchk --help`.) With no options, `isamchk` simply checks your table. To get more information or to tell `isamchk` to take corrective action, specify options as described below and in the following sections.

`tbl_name` is the database table you want to check. If you run `isamchk` somewhere other than in the database directory, you must specify the path to the file, since `isamchk` has no idea where your database is located. Actually, `isamchk` doesn't care whether or not the files you are working on are located in a database directory; you can copy the files that

correspond to a database table into another location and perform recovery operations on them there.

You can name several tables on the `isamchk` command line if you wish. You can also specify a name as an index file name (with the '`.ISM`' suffix), which allows you to specify all tables in a directory by using the pattern '`*.ISM`'. For example, if you are in a database directory, you can check all the tables in the directory like this:

```
shell> isamchk *.ISM
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> isamchk /path/to/database_dir/*.ISM
```

You can even check all tables in all databases by specifying a wildcard with the path to the **MySQL** data directory:

```
shell> isamchk /path/to/datadir/*/*.ISM
```

`isamchk` supports the following options:

`-a, --analyze`
: Analyze the distribution of keys. This improves join performance by enabling the join optimizer to better choose in which order it should join the tables and which keys it should how use.

`-#, --debug=debug_options`
: Output debug log. The `debug_options` string often is `'d:t:o,filename'`.

`-d, --description`
: Prints some information about the table.

`-e, --extend-check`
: Check the table VERY thoroughly. This is necessary only in extreme cases. Normally, `isamchk` should find all errors even without this option.

`-f, --force`
: Overwrite old temporary files. If you use `-f` when checking tables (running `isamchk` without `-r`), `isamchk` will automatically restart with `-r` on any table for which an error occurs during checking.

`--help` Display a help message and exit.

`-i, --information`
: Print informational statistics about the table that is checked.

`-k #, --keys-used=#`
: Used with `-r`. Tell the NISAM table handler to update only the first `#` indexes. Higher-numbered indexes are deactivated. This can be used to get faster inserts! Deactivated indexes can be reactivated by using `isamchk -r`.

`-l, --no-symlinks`
: Do not follow symbolic links when repairing. Normally `isamchk` repairs the table a symlink points at.

`-q, --quick`
: Used with `-r` to get a faster repair. Normally, the original data file isn't touched; you can specify a second `-q` to force the original data file to be used.

`-r, --recover`
: Recovery mode. Can fix almost anything except unique keys that aren't unique.

`-o, --safe-recover`
: Recovery mode. Uses an old recovery method; this is slower than `-r`, but can handle a couple of cases that `-r` cannot handle.

`-O var=option, --set-variable var=option`
: Set the value of a variable. The possible variables are listed below.

`-s, --silent`
: Silent mode. Write output only when errors occur. You can use `-s` twice (`-ss`) to make `isamchk` very silent.

`-S, --sort-index`
: Sort the index tree blocks in high-low order. This will optimize seeks and will make table scanning by key faster.

`-R index_num, --sort-records=index_num`
: Sorts records according to an index. This makes your data much more localized and may speed up ranged `SELECT` and `ORDER BY` operations on this index. (It may be VERY slow to do a sort the first time!) To find out a table's index numbers, use `SHOW INDEX`, which shows a table's indexes in the same order that `isamchk` sees them. Indexes are numbered beginning with 1.

`-u, --unpack`
: Unpack a table that was packed with `pack_isam`.

`-v, --verbose`
: Verbose mode. Print more information. This can be used with `-d` and `-e`. Use `-v` multiple times (`-vv`, `-vvv`) for more verbosity!

`-V, --version`
: Print the `isamchk` version and exit.

`-w, --wait`
: Wait if the table is locked.

Possible variables for the `--set-variable` (`-O`) option are:

```
key_buffer_size       current value: 16776192
read_buffer_size      current value: 262136
write_buffer_size     current value: 262136
sort_buffer_size      current value: 2097144
sort_key_blocks       current value: 16
decode_bits           current value: 9
```

13.1.2 `isamchk` memory usage

Memory allocation is important when you run `isamchk`. `isamchk` uses no more memory than you specify with the `-O` options. If you are going to use `isamchk` on very large files, you should first decide how much memory you want it to use. The default is to use only about 3M to fix things. By using larger values, you can get `isamchk` to operate faster. For example, if you have more than 32M RAM, you could use options such as these (in addition to any other options you might specify):

```
shell> isamchk -O sort=16M -O key=16M -O read=1M -O write=1M ...
```

Using `-O sort=16M` should probably be enough for most cases.

Be aware that `isamchk` uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory file system, you may easily get out of memory errors. If this happens, set `TMPDIR` to point at some directory with more space and restart `isamchk`

13.2 Setting up a table maintenance regimen

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. For maintenance purposes, you can use `isamchk -s` to check tables. The `-s` option causes `isamchk` to run in silent mode, printing messages only when errors occur.

It's a good idea to check tables when the server starts up. For example, whenever the machine has done a reboot in the middle of an update, you usually need to check all the tables that could have been affected. (This is an "expected crashed table".) You could add a test to `safe_mysqld` that runs `isamchk` to check all tables that have been modified during the last 24 hours if there is an old '`.pid`' (process ID) file left after a reboot. (The '`.pid`' file is created by `mysqld` when it starts up and removed when it terminates normally. The presence of a '`.pid`' file at system startup time indicates that `mysqld` terminated abnormally.)

An even better test would be to check any table whose last-modified time is more recent than that of the '`.pid`' file.

You should also check your tables regularly during normal system operation. At TcX, we run a `cron` job to check all our important tables once a week, using a line like this in a '`crontab`' file:

```
35 0 * * 0 /path/to/isamchk -s /path/to/datadir/*/*.ISM
```

This prints out information about crashed tables so we can examine and repair them when needed.

As we haven't had any unexpectedly crashed tables (tables that become corrupted for reasons other than hardware trouble) for a couple of years now (this is really true), once a week is more than enough for us.

We recommend that to start with, you execute `isamchk -s` each night on all tables that have been updated during the last 24 hours, until you come to trust **MySQL** as much as we do.

13.3 Getting information about a table

To get a description of a table or statistics about it, use the commands shown below. We explain some of the information in more detail later.

`isamchk -d tbl_name`
: Runs `isamchk` in "describe mode" to produce a description of your table. If you start the **MySQL** server using the `--skip-locking` option, `isamchk` may report an error for a table that is updated while it runs. However, since `isamchk` doesn't change the table in describe mode, there isn't any risk of destroying data.

`isamchk -d -v tbl_name`
: To produce more information about what `isamchk` is doing, add `-v` to tell it to run in verbose mode.

`isamchk -eis tbl_name`
: Shows only the most important information from a table. It is slow since it must read the whole table.

`isamchk -eiv tbl_name`
: This is like `-eis`, but tells you what is being done.

Example of `isamchk -d` output:

```
ISAM file:     company.ISM
Data records:           1403698  Deleted blocks:         0
Recordlength:               226
Record format: Fixed length

table description:
Key Start Len Index   Type
1   2     8   unique  double
2   15    10  multip. text packed stripped
3   219   8   multip. double
4   63    10  multip. text packed stripped
5   167   2   multip. unsigned short
6   177   4   multip. unsigned long
7   155   4   multip. text
8   138   4   multip. unsigned long
9   177   4   multip. unsigned long
    193   1           text
```

Example of `isamchk -d -v` output:

```
ISAM file:     company.ISM
Isam-version:  2
Creation time: 1996-08-28 11:44:22
Recover time:  1997-01-12 18:35:29
Data records:           1403698  Deleted blocks:              0
Datafile: Parts:        1403698  Deleted data:                0
Datafilepointer (bytes):      3  Keyfile pointer (bytes):     3
Max datafile length: 3791650815  Max keyfile length: 4294967294
```

```
Recordlength:               226
Record format: Fixed length

table description:
Key Start Len Index   Type                     Root Blocksize Rec/key
1   2     8   unique  double                15845376      1024       1
2   15    10  multip. text packed stripped  25062400      1024       2
3   219   8   multip. double                40907776      1024      73
4   63    10  multip. text packed stripped  48097280      1024       5
5   167   2   multip. unsigned short        55200768      1024    4840
6   177   4   multip. unsigned long         65145856      1024    1346
7   155   4   multip. text                  75090944      1024    4995
8   138   4   multip. unsigned long         85036032      1024      87
9   177   4   multip. unsigned long         96481280      1024     178
    193   1           text
```

Example of `isamchk -eis` output:

```
Checking ISAM file: company.ISM
Key:  1:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
Key:  2:  Keyblocks used:  98%  Packed:   50%  Max levels:  4
Key:  3:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
Key:  4:  Keyblocks used:  99%  Packed:   60%  Max levels:  3
Key:  5:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  6:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  7:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  8:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
Key:  9:  Keyblocks used:  98%  Packed:    0%  Max levels:  4
Total:    Keyblocks used:  98%  Packed:   17%

Records:          1403698    M.recordlength:     226   Packed:             0%
Recordspace used:     100%   Empty space:          0%  Blocks/Record:   1.00
Recordblocks:     1403698    Deleteblocks:         0
Recorddata:     317235748    Deleted data:         0
Lost space:             0    Linkdata:             0

User time 1626.51, System time 232.36
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 627, Swaps 0
Blocks in 0 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 639, Involuntary context switches 28966
```

Example of `isamchk -eiv` output:

```
Checking ISAM file: company.ISM
Data records: 1403698   Deleted blocks:       0
- check file-size
- check delete-chain
index  1:
index  2:
index  3:
```

```
index  4:
index  5:
index  6:
index  7:
index  8:
index  9:
No recordlinks
- check index reference
- check data record references index: 1
Key:  1:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
- check data record references index: 2
Key:  2:  Keyblocks used:  98%  Packed:   50%  Max levels:  4
- check data record references index: 3
Key:  3:  Keyblocks used:  97%  Packed:    0%  Max levels:  4
- check data record references index: 4
Key:  4:  Keyblocks used:  99%  Packed:   60%  Max levels:  3
- check data record references index: 5
Key:  5:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 6
Key:  6:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 7
Key:  7:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 8
Key:  8:  Keyblocks used:  99%  Packed:    0%  Max levels:  3
- check data record references index: 9
Key:  9:  Keyblocks used:  98%  Packed:    0%  Max levels:  4
Total:    Keyblocks used:   9%  Packed:   17%

- check records and index references
[LOTS OF ROW NUMBERS DELETED]

Records:          1403698    M.recordlength:     226   Packed:             0%
Recordspace used:     100%   Empty space:          0%  Blocks/Record:   1.00
Recordblocks:     1403698    Deleteblocks:         0
Recorddata:     317235748    Deleted data:         0
Lost space:             0    Linkdata:             0

User time 1639.63, System time 251.61
Maximum resident set size 0, Integral resident set size 0
Non physical pagefaults 0, Physical pagefaults 10580, Swaps 0
Blocks in 4 out 0, Messages in 0 out 0, Signals 0
Voluntary context switches 10604, Involuntary context switches 122798
```

Here are the sizes of the data and index files for the table used in the preceding examples:

```
-rw-rw-r--   1 monty    tcx     317235748 Jan 12 17:30 company.ISD
-rw-rw-r--   1 davida   tcx      96482304 Jan 12 18:35 company.ISM
```

Explanations for the types of information `isamchk` produces are given below. The "keyfile" is the index file. "Record" and "row" are synonymous.

`ISAM file` Name of the ISAM (index) file.

`Isam-version`
: Version of ISAM format. Currently always 2.

`Creation time`
: When the data file was created.

`Recover time`
: When the index/data file was last reconstructed.

`Data records`
: How many records are in the table.

`Deleted blocks`
: How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 13.4.3 [Optimization], page 292.

`Datafile: Parts`
: For dynamic record format, this indicates how many data blocks there are. For an optimized table without fragmented records, this is the same as `Data records`.

`Deleted data`
: How many bytes of non-reclaimed deleted data there are. You can optimize your table to minimize this space. See Section 13.4.3 [Optimization], page 292.

`Datafile pointer`
: The size of the data file pointer, in bytes. It is usually 2, 3, 4 or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from **MySQL** yet. For fixed tables, this is a record address. For dynamic tables, this is a byte address.

`Keyfile pointer`
: The size of the index file pointer, in bytes. It is usually 1, 2 or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by **MySQL**. It is always a block address.

`Max datafile length`
: How long the table's data file (`.ISD` file) can become, in bytes.

`Max keyfile length`
: How long the table's key file (`.ISM` file) can become, in bytes.

`Recordlength`
: How much space each record takes, in bytes.

`Record format`
: The format used to store table rows. The examples shown above use `Fixed length`. Other possible values are `Compressed` and `Packed`.

`table description`
: A list of all keys in the table. For each key, some low-level information is presented:

`Key` This key's number.

`Start` Where in the record this index part starts.

`Len` How long this index part is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column.

`Index` `unique` or `multip.` (multiple). Indicates whether or not one value can exist multiple times in this index.

`Type` What data-type this index part has. This is an NISAM data-type with the options `packed`, `stripped` or `empty`.

`Root` Address of the root index block.

`Blocksize`
The size of each index block. By default this is 1024, but the value may be changed at compile time.

`Rec/key` This is a statistical value used by the optimizer. It tells how many records there are per value for this key. A unique key always has a value of 1. This may be updated after a table is loaded (or greatly changed) with `isamchk -a`. If this is not updated at all, a default value of 30 is given.

In the first example above, the 9th key is a multi-part key with two parts.

`Keyblocks used`
What percentage of the keyblocks are used. Since the table used in the examples had just been reorganized with `isamchk`, the values are very high (very near the theoretical maximum).

`Packed` **MySQL** tries to pack keys with a common suffix. This can only be used for `CHAR/VARCHAR/DECIMAL` keys. For long strings like names, this can significantly reduce the space used. In the third example above, the 4th key is 10 characters long and a 60% reduction in space is achieved.

`Max levels`
How deep the B-tree for this key is. Large tables with long keys get high values.

`Records` How many rows are in the table.

`M.recordlength`
The average record length. For tables with fixed-length records, this is the exact record length.

`Packed` **MySQL** strips spaces from the end of strings. The `Packed` value indicates the percentage savings achieved by doing this.

`Recordspace used`
What percentage of the data file is used.

`Empty space`
What percentage of the data file is unused.

`Blocks/Record`
: Average number of blocks per record (i.e., how many links a fragmented record is composed of). This is always 1 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too big, you can reorganize the table with `isamchk`. See Section 13.4.3 [Optimization], page 292.

`Recordblocks`
: How many blocks (links) are used. For fixed format, this is the same as the number of records.

`Deleteblocks`
: How many blocks (links) are deleted.

`Recorddata`
: How many bytes in the data file are used.

`Deleted data`
: How many bytes in the data file are deleted (unused).

`Lost space`
: If a record is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

`Linkdata`
: When the dynamic table format is used, record fragments are linked with pointers (4 to 7 bytes each). `Linkdata` is the sum of the amount of storage used by all such pointers.

If a table has been compressed with `pack_isam`, `isamchk -d` prints additional information about each table column. See Section 12.5 [pack_isam], page 272, for an example of this information and a description of what it means.

13.4 Using `isamchk` for crash recovery

The file format that **MySQL** uses to store data has been extensively tested, but there are always external circumstances that may cause database tables to become corrupted:

- The `mysqld` process being killed in the middle of a write
- Unexpected shutdown of the computer (for example, if the computer is turned off)
- A hardware error

This chapter describes how to check for and deal with data corruption in **MySQL** databases. If your tables get corrupted a lot you should try to find the reason for this! See Section G.1 [Debugging server], page 444.

When performing crash recovery, it is important to understand that each table `tbl_name` in a database corresponds to three files in the database directory:

File	Purpose
'`tbl_name.frm`'	Table definition (form) file
'`tbl_name.ISD`'	Data file
'`tbl_name.ISM`'	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

`isamchk` works by creating a copy of the '`.ISD`' (data) file row by row. It ends the repair stage by removing the old '`.ISD`' file and renaming the new file to the original file name. If you use `--quick`, `isamchk` does not create a temporary '`.ISD`' file, but instead assumes that the '`.ISD`' file is correct and only generates a new index file without touching the '`.ISD`' file. This is safe, because `isamchk` automatically detects if the '`.ISD`' file is corrupt and aborts the repair in this case. You can also give two `--quick` options to `isamchk`. In this case, `isamchk` does not abort on some errors (like duplicate key) but instead tries to resolve them by modifying the '`.ISD`' file. Normally the use of two `--quick` options is useful only if you have too little free disk space to perform a normal repair. In this case you should at least make a backup before running `isamchk`.

13.4.1 How to check tables for errors

To check a table, use the following commands:

`isamchk tbl_name`
: This finds 99.99% of all errors. What it can't find is corruption that involves **ONLY** the data file (which is very unusual). If you want to check a table, you should normally run `isamchk` without options or with either the `-s` or `--silent` option.

`isamchk -e tbl_name`
: This does a complete and thorough check of all data (`-e` means "extended check"). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a LONG time on a big table with many keys. `isamchk` will normally stop after the first error it finds. If you want to obtain more information, you can add the `--verbose` (`-v`) option. This causes `isamchk` to keep going, up through a maximum of 20 errors. In normal usage, a simple `isamchk` (with no arguments other than the table name) is sufficient.

`isamchk -e -i tbl_name`
: Like the previous command, but the `-i` option tells `isamchk` to print some informational statistics, too.

13.4.2 How to repair tables

The symptoms of a corrupted table are usually that queries abort unexpectedly and that you observe errors such as these:

- '`tbl_name.frm`' is locked against change
- Can't find file '`tbl_name.ISM`' (Errcode: ###)
- Got error ### from table handler (Error 135 is an exception in this case)
- Unexpected end of file
- Record file is crashed

In these cases, you must repair your tables. `isamchk` can usually detect and fix most things that go wrong.

The repair process involves up to four stages, described below. Before you begin, you should `cd` to the database directory and check the permissions of the table files. Make sure they are readable by the Unix user that `mysqld` runs as (and to you, since you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

Stage 1: Checking your tables

Run `isamchk *.ISM` or (`isamchk -e *.ISM` if you have more time). Use the `-s` (silent) option to suppress unnecessary information.

You have to repair only those tables for which `isamchk` announces an error. For such tables, proceed to Stage 2.

If you get weird errors when checking (such as `out of memory` errors), or if `isamchk` crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try `isamchk -r -q tbl_name` (`-r -q` means "quick recovery mode"). This will attempt to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use `isamchk -r tbl_name` (`-r` means "recovery mode"). This will remove incorrect records and deleted records from the data file and reconstruct the index file.
3. If the preceding step fails, use `isamchk --safe-recover tbl_name`. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode doesn't (but is slower).

If you get weird errors when repairing (such as `out of memory` errors), or if `isamchk` crashes, go to Stage 3.

Stage 3: Difficult repair

You should only reach this stage if the first 16K block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it's necessary to create a new index file. Do so as follows:

1. Move the data file to some safe place.
2. Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
mysql> DELETE FROM tbl_name;
mysql> quit
```

3. Copy the old data file back onto the newly created data file. (Don't just move the old file back onto the new file; you want to retain a copy in case something goes wrong.)

Go back to Stage 2. `isamchk -r -q` should work now. (This shouldn't be an endless loop).

Stage 4: Very difficult repair

You should reach this stage only if the description file has also crashed. That should never happen, because the description file isn't changed after the table is created.

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with `isamchk -r`.
2. If you don't have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, then move the description and index files from the other database to your crashed database. This gives you new description and index files, but leaves the data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

13.4.3 Table optimization

To coalesce fragmented records and eliminate wasted space resulting from deleting or updating records, run `isamchk` in recovery mode:

```
shell> isamchk -r tbl_name
```

You can optimize a table in the same way using the SQL `OPTIMIZE TABLE` statement. `OPTIMIZE TABLE` is easier, but `isamchk` is faster.

`isamchk` also has a number of other options you can use to improve the performance of a table:

`-S, --sort-index`

`-R index_num, --sort-records=index_num`

`-a, --analyze`

For a full description of the option see Section 13.1.1 [isamchk syntax], page 280.

13.5 Log file maintenance

When using **MySQL** with log files, you will from time to time want to remove/backup old log files and tell **MySQL** to start logging on new files. See Section 9.2 [Update log], page 242.

One a Linux (`Redhat`) installation, you can use the `mysql-log-rotate` script for this. If you installed **MySQL** from an RPM distribution, the script should have been installed automatically.

On other systems you must install a short script yourself that you start from `cron` to handle log files.

You can force **MySQL** to start using new log files by using `mysqladmin flush-logs` or by using the SQL command `FLUSH LOGS`. If you are using **MySQL** 3.21 you must use `mysqladmin refresh`.

The above command does the following:

- If standard logging (`--log`) is used, closes and reopens the log file. ('`mysql.log`' as default).
- If update logging (`--log-update`) is used, closes the update log and opens a new log file with a higher sequence number.

If you are using only an update log, you only have to flush the logs and then move away the old update log files to a backup. If you are using the normal logging, you can do something like:

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mysqladmin flush-tables
```

and then take a backup and remove '`mysql.old`'.

14 Adding new functions to MySQL

There are two ways to add new functions to **MySQL**:

- You can add the function through the user-definable function (UDF) interface. User-definable functions are added and removed dynamically using the `CREATE FUNCTION` and `DROP FUNCTION` statements. See Section 7.29 [CREATE FUNCTION], page 205.
- You can add the function as a native (built in) **MySQL** function. Native functions are compiled into the `mysqld` server and become available on a permanent basis.

Each method has advantages and disadvantages:

- If you write a user-definable function, you must install the object file in addition to the server itself. If you compile your function into the server, you don't need to do that.
- You can add UDFs to a binary **MySQL** distribution. Native functions require you to modify a source distribution.
- If you upgrade your **MySQL** distribution, you can continue to use your previously-installed UDFs. For native functions, you must repeat your modifications each time you upgrade.

Whichever method you use to add new functions, they may be used just like native functions such as `ABS()` or `SOUNDEX()`.

14.1 Adding a new user-definable function

For the UDF mechanism to work, functions must be written in C or C++ and your operating system must support dynamic loading. The **MySQL** source distribution includes a file '`sql/udf_example.cc`' that defines 5 new functions. Consult this file to see how UDF calling conventions work.

For each function that you want to use in SQL statements, you should define corresponding C (or C++) functions. In the discussion below, the name "xxx" is used for an example function name. To distinquish between SQL and C/C++ usage, `XXX()` (uppercase) indicates a SQL function call, and `xxx()` (lowercase) indicates a C/C++ function call.

The C/C++ functions that you write to implement the inferface for `XXX()` are:

`xxx()` (required)
: The main function. This is where the function result is computed. The correspondence between the SQL type and return type of your C/C++ function is shown below:

SQL type	C/C++ type
`STRING`	`char *`
`INTEGER`	`long long`
`REAL`	`double`

`xxx_init()` (optional)
: The initialization function for `xxx()`. It can be used to:

  - Check the number of arguments to `XXX()`

- Check that the arguments are of a required type, or, alternatively, tell **MySQL** to coerce arguments to the types you want when the main function is called
- Allocate any memory required by the main function
- Specify the maximum length of the result
- Specify (for `REAL` functions) the maximum number of decimals
- Specify whether or not the result can be `NULL`

`xxx_deinit()` (optional)
: The deinitialization function for `xxx()`. It should deallocate any memory allocated by the initialization function.

When a SQL statement invokes `XXX()`, **MySQL** calls the initialization function `xxx_init()` to let it perform any required setup, such as argument checking or memory allocation. If `xxx_init()` returns an error, the SQL statement is aborted with an error message and the main and deinitialization functions are not called. Otherwise, the main function `xxx()` is called once for each row. After all rows have been processed, the deinitialization function `xxx_deinit()` is called so it can perform any required cleanup.

All functions must be thread-safe (not just the main function, but the initialization and deinitialization functions as well). This means that you are not allowed to allocate any global or static variables that change! If you need memory, you should allocate it in `xxx_init()` and free it in `xxx_deinit()`.

14.1.1 UDF calling sequences

The main function should be declared as shown below. Note that the return type and parameters differ, depending on whether you will declare the SQL function `XXX()` to return `STRING`, `INTEGER` or `REAL` in the `CREATE FUNCTION` statement:

For `STRING` functions:

```
char *xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *result, unsigned long *length,
              char *is_null, char *error);
```

For `INTEGER` functions:

```
long long xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

For `REAL` functions:

```
double xxx(UDF_INIT *initid, UDF_ARGS *args,
              char *is_null, char *error);
```

The initialization and deinitialization functions are declared like this:

```
my_bool xxx_init(UDF_INIT *initid, UDF_ARGS *args, char *message);

void xxx_deinit(UDF_INIT *initid);
```

The `initid` parameter is passed to all three functions. It points to a `UDF_INIT` structure that is used to communicate information between functions. The `UDF_INIT` structure members

are listed below. The initialization function should fill in any members that it wishes to change. (To use the default for a member, leave it unchanged.)

`my_bool maybe_null`
: `xxx_init()` should set `maybe_null` to `1` if `xxx()` can return `NULL`. The default value is `1` if any of the arguments are declared `maybe_null`.

`unsigned int decimals`
: Number of decimals. The default value is the maximum number of decimals in the arguments passed to the main function. (For example, if the function is passed `1.34`, `1.345` and `1.3`, the default would be 3, since `1.345` has 3 decimals.

`unsigned int max_length`
: The maximum length of the string result. The default value differs depending on the result type of the function. For string functions, the default is the length of the longest argument. For integer functions, the default is 21 digits. For real functions, the default is 13 plus the number of decimals indicated by `initid->decimals`. (For numeric functions, the length includes any sign or decimal point characters.)

`char *ptr`
: A pointer that the function can use for its own purposes. For example, functions can use `initid->ptr` to communicate allocated memory between functions. In `xxx_init()`, allocate the memory and assign it to this pointer:

```
initid->ptr = allocated_memory;
```

In `xxx()` and `xxx_deinit()`, refer to `initid->ptr` to use or deallocate the memory.

14.1.2 Argument processing

The `args` parameter points to a `UDF_ARGS` structure which has the members listed below:

`unsigned int arg_count`
: The number of arguments. Check this value in the initialization function if you want your function to be called with a particular number of arguments. For example:

```
if (args->arg_count != 2)
{
    strcpy(message,"XXX() requires two arguments");
    return 1;
}
```

`enum Item_result *arg_type`
: The types for each argument. The possible type values are `STRING_RESULT`, `INT_RESULT` and `REAL_RESULT`.

To make sure that arguments are of a given type and return an error if they are not, check the `arg_type` array in the initialization function. For example:

```
if (args->arg_type[0] != STRING_RESULT
      && args->arg_type[1] != INT_RESULT)
{
```

```
        strcpy(message,"XXX() requires a string and an integer");
        return 1;
    }
```

As an alternative to requiring your function's arguments to be of particular types, you can use the initialization function to set the `arg_type` elements to the types you want. This causes **MySQL** to coerce arguments to those types for each call to `xxx()`. For example, to specify coercion of the first two arguments to string and integer, do this in `xxx_init()`:

```
args->arg_type[0] = STRING_RESULT;
args->arg_type[1] = INT_RESULT;
```

`char **args`

`args->args` communicates information to the initialization function about the general nature of the arguments your function was called with. For a constant argument `i`, `args->args[i]` points to the argument value. (See below for instructions on how to access the value properly.) For a non-constant argument, `args->args[i]` is `0`. A constant argument is an expression that uses only constants, such as `3` or `4*7-2` or `SIN(3.14)`. A non-constant argument is an expression that refers to values that may change from row to row, such as column names or functions that are called with non-constant arguments.

For each invocation of the main function, `args->args` contains the actual arguments that are passed for the row currently being processed.

Functions can refer to an argument `i` as follows:

- An argument of type `STRING_RESULT` is given as a string pointer plus a length, to allow handling of binary data or data of arbitrary length. The string contents are available as `args->args[i]` and the string length is `args->lengths[i]`. You should not assume that strings are null-terminated.
- For an argument of type `INT_RESULT`, you must cast `args->args[i]` to a `long long` value:

  ```
  long long int_val;
  int_val = *((long long*) args->args[i]);
  ```

- For an argument of type `REAL_RESULT`, you must cast `args->args[i]` to a `double` value:

  ```
  double    real_val;
  real_val = *((double*) args->args[i]);
  ```

`unsigned long *lengths`

For the initialization function, the `lengths` array indicates the maximum string length for each argument. For each invocation of the main function, `lengths` contains the actual lengths of any string arguments that are passed for the row currently being processed. For arguments of types `INT_RESULT` or `REAL_RESULT`, `lengths` still contains the maximum length of the argument (as for the initialization function).

14.1.3 Return values and error handling

The initialization function should return `0` if no error occurred and `1` otherwise. If an error occurs, `xxx_init()` should store a null-terminated error message in the `message` parameter. The message will be returned to the client. The message buffer is `MYSQL_ERRMSG_SIZE` characters long, but you should try to keep the message to less than 80 characters so that it fits the width of a standard terminal screen.

The return value of the main function `xxx()` is the function value, for `long long` and `double` functions. For string functions, the string is returned in the `result` and `length` arguments. `result` is a buffer at least 255 bytes long. Set these to the contents and length of the return value. For example:

```
memcpy(result, "result string", 13);
*length = 13;
```

The string function return value normally also points to the result.

To indicate a return value of `NULL` in the main function, set `is_null` to `1`:

```
*is_null = 1;
```

To indicate an error return in the main function, set the `error` parameter to `1`:

```
*error = 1;
```

If `xxx()` sets `*error` to `1` for any row, the function value is `NULL` for the current row and for any subsequent rows processed by the statement in which `XXX()` was invoked. (`xxx()` will not even be called for subsequent rows.) **Note:** In **MySQL** versions prior to 3.22.10, you should set both `*error` and `*is_null`:

```
*error = 1;
*is_null = 1;
```

14.1.4 Compiling and installing user-definable functions

Files implementing UDFs must be compiled and installed on the host where the server runs. This process is described below for the example UDF file '`udf_example.cc`' that is included in the **MySQL** source distribution. This file contains the following functions:

- `metaphon()` returns a metaphon string of the string argument. This is something like a soundex string, but it's more tuned for English.
- `myfunc_double()` returns the sum of the ASCII values of the characters in its arguments, divided by the sum of the length of its arguments.
- `myfunc_int()` returns the sum of the length of its arguments.
- `lookup()` returns the IP number for a hostname.
- `reverse_lookup()` returns the hostname for an IP number. The function may be called with a string `"xxx.xxx.xxx.xxx"` or four numbers.

A dynamically-loadable file should be compiled as a sharable object file, using a command something like this:

```
shell> gcc -shared -o udf_example.so myfunc.cc
```

You can easily find out the correct compiler options for your system by running this command in the '`sql`' directory of your **MySQL** source tree:

```
shell> make udf_example.o
```

You should run a compile command similar to the one that `make` displays, except that you should remove the `-c` option near the end of the line and add `-o udf_example.so` to the end of the line. (On some systems, you may need to leave the `-c` on the command.)

Once you compile a shared object containing UDFs, you must install it and tell **MySQL** about it. Compiling a shared object from '`udf_example.cc`' produces a file named something like '`udf_example.so`' (the exact name may vary from platform to platform). Copy this file to some directory searched by `ld`, such as '`/usr/lib`'. On many systems, you can set the `LD_LIBRARY` or `LD_LIBRARY_PATH` environment variable to point at the directory where you have your UDF function files. The `dopen` manual page tells you which variable you should use on your system. You should set this in `mysql.server` or `safe_mysqld` and restart `mysqld`.

After the library is installed, notify `mysqld` about the new functions with these commands:

```
mysql> CREATE FUNCTION metaphon RETURNS STRING SONAME "udf_example.so";
mysql> CREATE FUNCTION myfunc_double RETURNS REAL SONAME "udf_example.so";
mysql> CREATE FUNCTION myfunc_int RETURNS INTEGER SONAME "udf_example.so";
mysql> CREATE FUNCTION lookup RETURNS STRING SONAME "udf_example.so";
mysql> CREATE FUNCTION reverse_lookup RETURNS STRING SONAME "udf_example.so";
```

Functions can be deleted using `DROP FUNCTION`:

```
mysql> DROP FUNCTION metaphon;
mysql> DROP FUNCTION myfunc_double;
mysql> DROP FUNCTION myfunc_int;
mysql> DROP FUNCTION lookup;
mysql> DROP FUNCTION reverse_lookup;
```

The `CREATE FUNCTION` and `DROP FUNCTION` statements update the system table `func` in the `mysql` database. The function's name, type and shared library name are saved in the table. You must have the **insert** and **delete** privileges for the `mysql` database to create and drop functions.

You should not use `CREATE FUNCTION` to add a function that has already been created. If you need to reinstall a function, you should remove it with `DROP FUNCTION` and then reinstall it with `CREATE FUNCTION`. You would need to do this, for example, if you recompile a new version of your function, so that `mysqld` gets the new version. Otherwise the server will continue to use the old version.

Active functions are reloaded each time the server starts, unless you start `mysqld` with the `--skip-grant-tables` option. In this case, UDF initialization is skipped and UDFs are unavailable. (An active function is one that has been loaded with `CREATE FUNCTION` and not removed with `DROP FUNCTION`.)

14.2 Adding a new native function

The procedure for adding a new native function is described below. Note that you cannot add native functions to a binary distribution since the procedure involves modifying **MySQL** source code. You must compile **MySQL** yourself from a source distribution. Also note that if you migrate to another version of **MySQL** (e.g., when a new version is released), you will need to repeat the procedure with the new version.

To add a new native **MySQL** function, follow these steps:

1. Add one line to '`lex.h`' that defines the function name in the `sql_functions[]` array.
2. Add two lines to '`sql_yacc.yy`'. One indicates the preprocessor symbol that `yacc` should define (this should be added at the beginning of the file). Then define the function parameters and add an "item" with these parameters to the `simple_expr` parsing rule. For an example, check all occurrences of `SOUNDEX` in '`sql_yacc.yy`' to see how this is done.
3. In '`item_func.h`', declare a class inheriting from `Item_num_func` or `Item_str_func`, depending on whether your function returns a number or a string.
4. In '`item_func.cc`', add one of the following declarations, depending on whether you are defining a numeric or string function:

```
double   Item_func_newname::val()
longlong Item_func_newname::val_int()
String  *Item_func_newname::Str(String *str)
```

5. You should probably also define the following function:

```
void Item_func_newname::fix_length_and_dec()
```

   This function should at least calculate `max_length` based on the given arguments. `max_length` is the maximum number of characters the function may return. This function should also set `maybe_null = 0` if the main function can't return a `NULL` value. The function can check if any of the function arguments can return `NULL` by checking the arguments `maybe_null` variable.

All functions must be thread-safe.

For string functions, there are some additional considerations to be aware of:

- The `String *str` argument provides a string buffer that may be used to hold the result.
- The function should return the string that holds the result.
- All current string functions try to avoid allocating any memory unless absolutely necessary!

15 Adding new procedures to MySQL

In **MySQL**, you can define a procedure in C++ that can access and modify the data in a query before sending it to a client. The modification can be done on row by row or `GROUP BY` level.

We have in **MySQL** 3.23 created an example procedure to show you what can be done.

15.1 Procedure analyse

`analyse([max elements,[max memory]])`

This procedure is defined in the '`sql/sql_analyse.cc`'. This examines the result from your query and returns an analysis of the results.

- `max elements` (default 256) is the maximum number of distinct values `analyse` will notice per column. This is used by `analyse` to check if the optimal column type should be of type `ENUM`.
- `max memory` (default 8192) is the maximum memory `analyse` should allocate per column while trying to find all distinct values.

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max elements,[max memory]])
```

15.2 Writing a procedure.

For the moment, the only documentation for this is the source. :(

You can find all information about procedures by examining the following files:

- '`sql/sql_analyse.cc`'
- '`sql/procedure.h`'
- '`sql/procedure.cc`'
- '`sql/sql_select.cc`'

16 MySQL ODBC Support

MySQL provides support for ODBC by means of the **MyODBC** program.

16.1 Operating systems supported by MyODBC

MyODBC is a 32-bit ODBC (2.50) level 0 driver for Windows95 and Windows NT. We hope somebody will port it to Windows 3.x.

16.2 How to report problems with MyODBC

MyODBC has been tested with Access, Admndemo.exe, C++-Builder, Centura Team Developer (formerly Gupta SQL/Windows), ColdFusion (on Solaris and NT with svc pack 5), Crystal Reports, DataJunction, Delphi, ERwin, Excel, iHTML, FileMaker Pro, FoxPro, Notes 4.5/4.6, SBSS, Perl DBD-ODBC, Paradox, Powerbuilder, Powerdesigner 32 bit, VC++ and Visual Basic.

If you know of any other application that works with **MyODBC**, please mail myodbc@lists.mysql.com about this!

If you encounter difficulties, we would like to have the log file from the ODBC manager (the log you get when requesting logs from ODBCADMIN) and a **MyODBC** log. This will help shed some light on any problems.

To get a **MyODBC** log, please tag the 'Trace MyODBC' option flag in the **MyODBC** connect/configure screen. The log will be written to file '`C:\myodbc.log`'. Note that you must use `MYSQL.DLL` and not `MYSQL2.DLL` for this option to work!

16.3 Programs known to work with MyODBC

Most programs should work with **MyODBC**, but for each of those listed below, we have tested it ourselves or gotten confirmation from some user that it works:

Program	Comment
Access	To make Access work: • You should have a primary key in the table. • You should have a timestamp in all tables you want to be able to update. • Only use double float fields. Access fails when comparing with single floats. • Set the 'Return matching rows' option field when connecting to **MySQL**. • Access on NT will report `BLOB` columns as `OLE OBJECTS`. If you want to have `MEMO` columns instead, you should change the column to `TEXT` with `ALTER TABLE`. • Access can't always handle `DATE` columns properly. If you have a problem with these, change the columns to `DATETIME`. • In some cases, Access may generate illegal SQL queries that **MySQL** can't understand. You can fix this by selecting `"Query|SQLSpecific|Pass-Through"` from the Access menu.

DataJunction
: You have to change it to output `VARCHAR` rather than `ENUM`, as it exports the latter in a manner that causes **MySQL** grief.

Excel
: Works. Some tips:

  - If you have problems with dates, try to select them as strings using the `CONCAT()` function. For example:

    ```
    select CONCAT(rise_time), CONCAT(set_time)
        from sunrise_sunset;
    ```

    Values retrieved as strings this way should be correctly recognized as time values by Excel97.

    The purpose of `CONCAT()` in this example is to fool ODBC into thinking the column is of "string type". Without the `CONCAT()`, ODBC knows the column is of time type, and Excel does not understand that.

    Note that this is a bug in Excel, because it automatically converts a string to a time. This would be great if the source was a text file, but is plain stupid when the source is an ODBC connection that reports exact types for each column.

odbcadmin
: Test program for ODBC.

Delphi
: You must use DBE 3.2 or newer. Set the 'Don't optimize column width' option field when connecting to **MySQL**.

  Also, here is some potentially useful delphi code that sets up both an ODBC entry and a BDE entry for MyODBC (the BDE entry requires a BDE Alias Editor which may be had for free at a Delphi Super Page near you.): (Thanks to Bryan Brunton bryan@flesherfab.com for this)

  ```
  fReg:= TRegistry.Create;
    fReg.OpenKey('\Software\ODBC\ODBC.INI\DocumentsFab', True);
    fReg.WriteString('Database', 'Documents');
    fReg.WriteString('Description', ' ');
    fReg.WriteString('Driver', 'C:\WINNT\System32\myodbc.dll');
    fReg.WriteString('Flag', '1');
    fReg.WriteString('Password', '');
    fReg.WriteString('Port', ' ');
    fReg.WriteString('Server', 'xmark');
    fReg.WriteString('User', 'winuser');
    fReg.OpenKey('\Software\ODBC\ODBC.INI\ODBC Data Sources', True);
    fReg.WriteString('DocumentsFab', 'MySQL');
    fReg.CloseKey;
    fReg.Free;

    Memo1.Lines.Add('DATABASE NAME=');
    Memo1.Lines.Add('USER NAME=');
    Memo1.Lines.Add('ODBC DSN=DocumentsFab');
    Memo1.Lines.Add('OPEN MODE=READ/WRITE');
  ```

```
Memo1.Lines.Add('BATCH COUNT=200');
Memo1.Lines.Add('LANGDRIVER=');
Memo1.Lines.Add('MAX ROWS=-1');
Memo1.Lines.Add('SCHEMA CACHE DIR=');
Memo1.Lines.Add('SCHEMA CACHE SIZE=8');
Memo1.Lines.Add('SCHEMA CACHE TIME=-1');
Memo1.Lines.Add('SQLPASSTHRU MODE=SHARED AUTOCOMMIT');
Memo1.Lines.Add('SQLQRYMODE=');
Memo1.Lines.Add('ENABLE SCHEMA CACHE=FALSE');
Memo1.Lines.Add('ENABLE BCD=FALSE');
Memo1.Lines.Add('ROWSET SIZE=20');
Memo1.Lines.Add('BLOBS TO CACHE=64');
Memo1.Lines.Add('BLOB SIZE=32');

AliasEditor.Add('DocumentsFab','MySQL',Memo1.Lines);
```

C++Builder

Tested with BDE 3.0. The only known problem is that when the table schema changes, query fields are not updated. BDE however does not seem to recognize primary keys, only the index PRIMARY, though this has not been a problem.

Visual basic

To be able to update a table, you must define a primary key for the table.

16.4 How to fill in the various fields in the ODBC administrator program

There are three possibilities for specifying the server name on Windows95:

- Use the IP address of the server.
- Add a file '`lmhosts`' with the following info:

  ```
  ip hostname
  ```

  For example:

  ```
  194.216.84.21 my
  ```

- Configure the PC to use DNS.

Example of how to fill in the "ODBC setup":

```
Windows DSN name:   test
Description:        This is my test database
MySql Database:     test
Server:             194.216.84.21
User:               monty
Password:           my_password
Port:
```

The value for the `Windows DSN name` field is any name that is unique in your windows ODBC setup.

You don't have to specify values for the `Server`, `User`, `Password` or `Port` fields in the ODBC setup screen. However, if you do, the values will be used as the defaults later when you attempt to make a connection. You have the option of changing the values at that time.

If the port number is not given, the default port (3306) is used.

If you specify the option `Read options from C:\my.cnf`, the groups `client` and `odbc` will be read from the '`C:\my.cnf`' file. You can use all options that are usable by `mysql_options()`. See Section 20.4.37 [mysql_options], page 354.

16.5 How to get the value of an AUTO_INCREMENT column in ODBC

A common problem is how to get the value of an automatically generated ID from an `INSERT`. With ODBC, you can do something like this (assuming that `auto` is an `AUTO_INCREMENT` field):

```
INSERT INTO foo (auto,text) VALUES(NULL,'text');
SELECT LAST_INSERT_ID();
```

Or, if you are just going to insert the ID into another table, you can do this:

```
INSERT INTO foo (auto,text) VALUES(NULL,'text');
INSERT INTO foo2 (id,text) VALUES(LAST_INSERT_ID(),'text');
```

For the benefit of some ODBC applications (at least Delphi and Access), the following query can be used to find a newly-inserted row:

```
SELECT * FROM tbl_name WHERE auto IS NULL;
```

17 Using MySQL with some common programs

17.1 Using MySQL with Apache

The contrib section includes programs that lets you authenticate your users from a **MySQL** database and also let you log your log files into a **MySQL** table. See Appendix B [Contrib], page 387.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

In **MySQL** you can now do something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE table_name
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

18 Problems and common errors

18.1 What to do if MySQL keeps crashing

All **MySQL** versions are tested on many platforms before they are released. This doesn't mean that there isn't any bugs in **MySQL**, but that if there are bugs they are very few and can be hard to find. If you have a problem, it will always help if you try to find out exactly what crashes your system as you will have a much better chance of getting this fixed quickly.

First you should try to find out whether the problem is that the `mysqld` daemon dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died, you may find the reason for this in the file '`mysql-data-directory/'hostname'.err`'.

Since it is very difficult to know why something is crashing, first try to check whether or not things that work for others crash for you. Please try the following things:

- Take down the `mysqld` daemon with `mysqladmin shutdown`, run `isamchk --silent --force */*.ISM` on all tables and restart the `mysqld` daemon. This will ensure that you are running from a clean state. See Chapter 13 [Maintenance], page 280.
- Use `mysqld --log` and try to determine from the information in the log whether or not some specific query kills the server. 95% of all bugs are related to a particular query! Normally this is one of the last queries in the log file just before **MySQL** restarted.

  You may be able to verify this using the following procedure:

  - Take down the **MySQL** daemon (with `mysqladmin shutdown`)
  - Make a backup of files in the **MySQL** database directory.
  - Check the tables with `isamchk -s */*.ISM` to verify that all tables are correct. If any table is corrupted, repair it with `isamchk -r path-to-table.ISM`.
  - Remove (or move away) any old log files from the **MySQL** data directory.
  - Start the server with `safe_mysql --log`.
  - If `mysqld` now dies, you can test if the problem is a specific query by restoring the backup and executing `mysql < mysql-log-file`. You can of course do the last test in some other directory than the standard **MySQL** database directory by starting another **MySQL** server with `safe_mysqld --data=path-to-backup-directory`.
- Have you tried the benchmarks? They should test **MySQL** rather well. You can also add code that simulates your application! The benchmarks can be found in the '`bench`' directory in the source distribution, or, for a binary distribution, in the '`sql-bench`' directory under your **MySQL** installation directory.
- Try `fork_test.pl` and `fork2_test.pl`.
- Check the file '`mysql-data-directory/'hostname'.err`' for any errors.
- If you configure **MySQL** for debugging, it will be much easier to gather information about possible errors if something goes wrong. Reconfigure **MySQL** with the `--with-debug` option to `configure` and then recompile. See Section G.1 [Debugging server], page 444.

- Configuring **MySQL** for debugging causes a safe memory allocator to be included that can find some errors. It also provides a lot of output about what is happening.
- Have you applied the latest patches for your operating system?
- Use the `--skip-locking` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-locking` option tells `mysqld` not to use external locking. (This means that you cannot run 2 `mysqld` servers on the same data and that you must be careful if you use `isamchk`, but it may be instructive to try the option as a test.)
- Have you tried `mysqladmin -u root processlist` when `mysqld` appears to be running but not responding? Sometimes `mysqld` is not comatose even though you might think so. The problem may be that all connections are in use, or there may be some internal lock problem. `mysqladmin processlist` will usually be able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.
- Run the command `mysqladmin -i 5 status` in a separate window to produce statistics while you run your other queries.
- Try the following:
  1. Start `mysqld` from `gdb` (or another debugger).
  2. Run your test scripts.
  3. Do `back` (or the backtrace command in your debugger) when `mysqld` core dumps.
- Try to simulate your application with a Perl script to force **MySQL** to crash or misbehave.
- Or send a normal bug report. See Section 2.3 [Bug reports], page 15. But be even more detailed than usual. Since **MySQL** works for many people, it may be that the crash results from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with table with dynamic length rows and you are not using `BLOB/TEXT` columns (but only `VARCHAR` columns) you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This will force **MySQL** to use fixed size rows. Fixed size rows take a little extra place, but are much more tolerant for corruption!

  The current dynamic row code has been in use at TCX for at least 3 years without any problems, but by nature dynamic length rows are more prone to errors so it may be a good idea to try if the above helps!

18.2 Some common errors when using MySQL

18.2.1 `MySQL server has gone away` error

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. By default, the server closes the connection after 8 hours if nothing has happened. You can change the time limit with by setting the `wait_timeout` variable when you start mysqld.

You can check that the **MySQL** hasn't died by executing `mysqladmin version` and examining the uptime.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection.

You normally can get the following error codes in this case (which one you get is OS-dependent):

`CR_SERVER_GONE_ERROR`	The client couldn't send a question to the server.
`CR_SERVER_LOST`	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` gets a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by starting `mysqld` with the `-O max_allowed_packet=#` option (default 1M). The extra memory is allocated on demand, so `mysqld` will use more memory only when you issue a big query or when `mysqld` must return a big result row!

18.2.2 `Can't connect to [local] MySQL server` error

A **MySQL** client can connect to the `mysqld` server in two different ways: Unix sockets, which connect through a file in the file system (default '`/tmp/mysqld.sock`'), or TCP/IP, which connects through a port number. Unix sockets are faster than TCP/IP but can only be used when connecting to a server on the same computer. Unix sockets are used if you don't specify a hostname or if you specify the special hostname `localhost`.

The error (2002) `Can't connect to ...` normally means that there isn't a **MySQL** server running on the system or that you are using a wrong socket file or TCP/IP port when trying to connect to the `mysqld` server.

Start by check (using `ps`) that there is a process running named `mysqld` on your server! If there isn't any `mysqld` process, you should start one. See Section 4.15.2 [Starting server], page 77

If a `mysqld` process is running, you can check the server by trying these different connections (the port number and socket pathname might be different in your setup, of course):

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h 'hostname' version variables
shell> mysqladmin -h 'hostname' --port=3306 version
shell> mysqladmin -h 'ip for your host' version
shell> mysqladmin --socket=/tmp/mysql.sock version
```

Note the use of backquotes rather than forward quotes with the `hostname` command; these cause the output of `hostname` (i.e., the current hostname) to be substituted into the `mysqladmin` command.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running.

- You are running on a system that uses MIT-pthreads. If you are running on a system that doesn't have native threads, `mysqld` uses the MIT-pthreads package. See Section 4.2 [Which OS], page 32. However, MIT-pthreads doesn't support Unix sockets, so on such a system you must always specify the hostname explicitly when connecting to the server. Try using this command to check the connection to the server:

```
shell> mysqladmin -h `hostname` version
```

- Someone has removed the Unix socket that `mysqld` uses (default '`/tmp/mysqld.sock`'). You might have a `cron` job that removes the **MySQL** socket (e.g., a job that removes old files from the '`/tmp`' directory). You can always run `mysqladmin version` and check that the socket `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove '`mysqld.sock`' or to place the socket somewhere else. You can specify a different socket location at **MySQL** configuration time with this command:

```
shell> ./configure --with-unix-socket-path=/path/to/socket
```

  You can also start `safe_mysqld` with the `--socket=/path/to/socket` option and set the environment variable `MYSQL_UNIX_PORT` to the socket pathname before starting your **MySQL** clients.
- You have started the `mysqld` server with the `--socket=/path/to/socket` option. If you change the socket pathname for the server, you must also notify the **MySQL** clients about the new path. You can do this by setting the environment variable `MYSQL_UNIX_PORT` to the socket pathname or by providing the socket path as an argument to the clients. You can test the socket with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

- You are using Linux and one thread has died (core dumped). In this case you must kill the other `mysqld` threads (for example with the `mysql_zap` script before you can start a new **MySQL** server. See Section 18.1 [Crashing], page 307

If you get the error message `Can't connect to MySQL server on some_hostname`, you can try the following things to find out what is the problem:

- Check if the server up by doing `telnet your-host-name tcp-ip-port-number` and press `RETURN` a couple of times. If there is a **MySQL** server running on this port you should get a responses that includes the version number of the running **MySQL** server. If you get an error like `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the used port.
- Try connecting to the `mysqld` daemon on the local machine and check the TCP/IP port that mysqld it's configured to use (variable `port`) with `mysqladmin variables`.
- Check that your `mysqld` server is not started with the `--skip-networking` option.

18.2.3 `Host '...' is blocked` error

If you get a error like this:

```
Host 'hostname' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

This means that `mysqld` has gotten a lot (`max_connect_errors`) of connect requests from the host `'hostname'` that have been interrupted in the middle. After `max_connect_errors`

failed requests, `mysqld` assumes that something is wrong (like a attack from a cracker), and blocks the site from further connections until someone executes the command `mysqladmin flush-hosts`.

By default, `mysqld` blocks a host after 10 connection errors. You can easily adjust this by starting the server like this:

```
shell> safe_mysqld -O max_connect_errors=10000 &
```

Note that if you get this error message for a given host, you should first check that there isn't anything wrong with TCP/IP connections from that host. If your TCP/IP connections aren't working, it won't do you any good to increase the value of the `max_connect_errors` variable!

18.2.4 `Out of memory` error

If you issue a query and get something like the following error:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

Note that the error refers to the **MySQL** client `mysql`. The reason for this error is simply that the client does not have enough memory to store the whole result.

To remedy the problem, first check that your query is correct. Is it reasonable that it should return so many rows? If so, you can use `mysql --quick`, which uses `mysql_use_result()` to retrieve the result set. This places less of a load on the client (but more on the server).

18.2.5 `Packet too large` error

When a **MySQL** client or the `mysqld` server gets a packet bigger than `max_allowed_packet` bytes, it issues a `Packet too large` error and closes the connection.

If you are using the `mysql` client, you may specify a bigger buffer by starting the client with `mysql --set-variable=max_allowed_packet=8M`.

If you are using other clients that do not allow you to specify the maximum packet size (such as `DBI`), you need to set the packet size when you start the server. You cau use a command-line option to `mysqld` to set `max_allowed_packet` to a larger size. For example, if you are expecting to store the full length of a `BLOB` into a table, you'll need to start the server with the `--set-variable=max_allowed_packet=24M` option.

18.2.6 `The table is full` error

This error occurs when an in-memory temporary table becomes larger than `tmp_table_size` bytes. To avoid this problem, you can use the `-O tmp_table_size=#` option to `mysqld` to increase the temporary table size, or use the SQL option `SQL_BIG_TABLES` before you issue the problematic query. See Section 7.24 [SET OPTION], page 199.

You can also start `mysqld` with the `--big-tables` option. This is exactly the same as using `SQL_BIG_TABLES` for all queries.

18.2.7 `Commands out of sync` error in client

If you get `Commands out of sync; You can't run this command now` in your client code, you are calling client functions in the wrong order!

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without a `mysql_use_result()` or `mysql_store_result()` in between.

18.2.8 `Ignoring user` error

If you get the following error:

`Found wrong password for user: 'some_user@some_host'; Ignoring user`

This means that when `mysqld` was started or when it reloaded the permissions tables, it found an entry in the `user` table with an invalid password. As a result, the entry is simply ignored by the permission system.

Possible causes of and fixes for this problem:

- You may be running a new version of `mysqld` with an old `user` table. You can check this by executing `mysqlshow mysql user` to see if the password field is shorter than 16 characters. If so, you can correct this condition by running the `scripts/add_long_password` script.
- The user has an old password (8 chararacters long) and you didn't start `mysqld` with the `--old-protocol` option. Update the user in the `user` table with a new password or restart `mysqld` with `--old-protocol`.
- You have specified a password in the `user` table without using the `PASSWORD()` function. Use `mysql` to update the user in the `user` table with a new password. Make sure to use the `PASSWORD()` function:

```
mysql> update user set password=PASSWORD('your password')
           where user='XXX';
```

18.2.9 `Table 'xxx' doesn't exist` error

If you get the error `Table 'xxx' doesn't exist` or `Can't find file: 'xxx' (errno: 2)`, this means that no table exists in the current database with the name `xxx`.

Note that as **MySQL** uses directories and files to store databases and tables, the database and table names are **case sensitive**! (On Win32 the databases and tables names are not case sensitive, but all references to a given table within a query must use the same case!)

You can check which tables you have in the current database with `SHOW TABLES`. See Section 7.20 [SHOW], page 190.

18.3 How MySQL handles a full disk

When a disk full condition occurs, **MySQL** does the following:

- It checks once every minute to see whether or not there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 6 minutes it writes an entry to the log file warning about the disk full condition.

To alleviate the problem, you can take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- To abort the thread, you must send a `mysqladmin kill` to the thread. The thread will be aborted the next time it checks the disk (in 1 minute).
- Note that other threads may be waiting for the table that caused the "disk full" condition. If you have several "locked" threads, killing the one thread that is waiting on the disk full condition will allow the other threads to continue.

18.4 How to run SQL commands from a text file

The `mysql` client typically is used interactively, like this:

```
shell> mysql database
```

However, it's also possible to put your SQL commands in a file and tell `mysql` to read its input from that file. To do so, create a text file '`text_file`' that contains the commands you wish to execute. Then invoke `mysql` as shown below:

```
shell> mysql database < text_file
```

You can also start your text file with a `USE db_name` statement. In this case, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

See Section 12.1 [Programs], page 266.

18.5 Where MySQL stores temporary files

MySQL uses the value of the `TMPDIR` environment variable as the pathname of the directory in which to store temporary files. If you don't have `TMPDIR` set, **MySQL** uses the system default, which is normally '`/tmp`' or '`/usr/tmp`'. If the file system containing your temporary file directory is too small, you should edit `safe_mysqld` to set `TMPDIR` to point to a directory in a file system where you have enough space! You can also set the temporary directory using the `--tmpdir` option to `mysqld`.

MySQL creates all temporary files as "hidden files". This ensures that the temporary files will be removed if `mysqld` is terminated. The disadvantage of using hidden files is that you will not see a big temporary file that fills up the file system in which the temporary file directory is located.

When sorting (`ORDER BY` or `GROUP BY`), **MySQL** normally uses one or two temporary files. The maximum disk-space needed is:

```
(length of what is sorted + sizeof(database pointer))
* number of matched rows
* 2
```

`sizeof(database pointer)` is usually 4, but may grow in the future for really big tables.

For some `SELECT` queries, **MySQL** also creates temporary SQL tables. These are not hidden and have names of the form '`SQL_*`'.

`ALTER TABLE` and `OPTIMIZE TABLE` create a temporary table in the same directory as the original table.

18.6 How to protect '`/tmp/mysql.sock`' from being deleted

If you have problems with the fact that anyone can delete the **MySQL** communication socket '`/tmp/mysql.sock`', you can, on most versions of Unix, protect your '`/tmp`' file system by setting the `sticky` bit on it. Log in as `root` and do the following:

```
shell> chmod +s /tmp
```

This will protect your '`/tmp`' file system so that files can be deleted only by their owners or the superuser (`root`).

You can check if the `sticky` bit is set by executing `ls -ld /tmp`. If the last permission bit is `t`, the bit is set.

18.7 `Access denied` error

See Section 6.6 [Privileges], page 98. And especially see Section 6.13 [Access denied], page 111.

18.8 How to run MySQL as a normal user

The **MySQL** server `mysqld` can be started and run by any user. In order to change `mysqld` to run as Unix user `user_name`, you must do the following:

1. Stop the server if it's running (use `mysqladmin shutdown`).
2. Change the database directories and files so that `user_name` has privileges to read and write files in them (you may need to do this as the Unix `root` user):

   ```
   shell> chown -R user_name /path/to/mysql/datadir
   ```

   If directories or files within the **MySQL** data directory are symlinks, you'll also need to follow those links and change the directories and files they point to. `chown -R` may not follow symlinks for you.
3. Start the server as user `user_name`, or, if you are using **MySQL** 3.22 or later, start `mysqld` as the Unix `root` user and use the `--user=user_name` option. `mysqld` will switch to run as Unix user `user_name` before accepting any connections.
4. If you are using the `mysql.server` script to start `mysqld` when the system is rebooted, you should edit `mysql.server` to use `su` to run `mysqld` as user `user_name`, or to invoke `mysqld` with the `--user` option. (No changes to `safe_mysqld` are necessary.)

At this point, your `mysqld` process should be running fine and dandy as the Unix user `user_name`. One thing hasn't changed, though: the contents of the permissions tables. By default (right after running the permissions table install script `mysql_install_db`), the **MySQL** user `root` is the only user with permission to access the `mysql` database or to create or drop databases. Unless you have changed those permissions, they still hold. This shouldn't stop

you from accessing **MySQL** as the **MySQL** `root` user when you're logged in as a Unix user other than `root`; just specify the `-u root` option to the client program.

Note that accessing **MySQL** as `root`, by supplying `-u root` on the command line, has *nothing* to do with **MySQL** running as the Unix `root` user, or, indeed, as other Unix user. The access permissions and user names of **MySQL** are completely separate from Unix user names. The only connection with Unix user names is that if you don't provide a `-u` option when you invoke a client program, the client will try to connect using your Unix login name as your **MySQL** user name.

If your Unix box itself isn't secured, you should probably at least put a password on the **MySQL** `root` users in the access tables. Otherwise, any user with an account on that machine can run `mysql -u root db_name` and do whatever he likes.

18.9 How to reset a forgotten password.

If you have forgot the root password for **MySQL**, you can restore this with the following procedure.

1. Take down the mysqld server by sending a `kill` (not `kill -9`) to the `mysqld` server. The pid is stored in a `.pid` file which is normally in the **MySQL** database directory:

   ```
   kill `cat /mysql-data-directory/hostname.pid`
   ```

2. Restart `mysqld` with the `--skip-grant-tables` option.
3. Connect to the mysqld server with `mysql -h hostname mysql` and change the password with a `GRANT` command. See Section 7.25 [GRANT], page 201. You can also do this with `mysqladmin -h hostname -u user password 'new password'`
4. Load the privilege tables with: `mysqladmin -h hostname flush-tables` or with the SQL command `FLUSH PRIVILEGES`.

18.10 Problems with file permissions

If you have problems with file permissions, for example, if `mysql` issues the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

Then the environment variable `UMASK` might be set incorrectly when `mysqld` starts up. The default umask value is `0660`. You can change this behavior by starting `safe_mysqld` as follows:

```
shell> UMASK=384  # = 600 in octal
shell> export UMASK
shell> /path/to/safe_mysqld &
```

18.11 File not found

If you get `ERROR '...' not found (errno: 23)`, `Can't open file: ... (errno: 24)` or any other error with `errno 23` or `errno 24` from **MySQL**, it means that you haven't allocated enough file descriptors for **MySQL**. You can use the `perror` utility to get a description of what the error number means:

```
shell> perror 23
File table overflow
shell> perror 24
Too many open files
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once, or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by using the `-O table_cache=32` option to `safe_mysqld` (the default value is 64). Reducing the value of `max_connections` will also reduce the number of open files (the default value is 90).

To change the number of file descriptors available to `mysqld`, modify the `safe_mysqld` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the `'#'` character to uncomment this line, and change the number 256 to change the number of file descriptors available to `mysqld`.

`ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. If you need to increase the OS limit on the number of file descriptors available to each process, consult the documentation for your operating system.

Note that if you run the `tcsh` shell, `ulimit` will not work! `tcsh` will also report incorrect values when you ask for the current limits! In this case you should start `safe_mysqld` with `sh`!

18.12 Problems using `DATE` columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to ANSI SQL, no other format is allowed. You should use this format in `UPDATE` expressions and in the WHERE clause of `SELECT` statements. For example:

```
mysql> SELECT * FROM tbl_name WHERE date >= '1997-05-05';
```

As a convenience, **MySQL** automatically converts a date to a number if the date is used in a numeric context (and vice versa). It is also smart enough to allow a "relaxed" string form when updating and in a `WHERE` clause that compares a date to a `TIMESTAMP`, `DATE` or a `DATETIME` column. (Relaxed form means that any non-numeric character may be used as the separator between parts. For example, `'1998-08-15'` and `'1998#08#15'` are equivalent.) **MySQL** can also convert a string containing no separators (such as `'19980815'`), provided it makes sense as a date.

The special date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'`. When using a `'0000-00-00'` date through **MyODBC**, it will automatically be converted to `NULL` in **MyODBC** 2.50.12 and above, because ODBC can't handle this kind of date.

Since **MySQL** performs the conversions described above, the following statements work:

```
mysql> INSERT INTO tbl_name (idate) VALUES (19970505);
mysql> INSERT INTO tbl_name (idate) VALUES ('19970505');
mysql> INSERT INTO tbl_name (idate) VALUES ('97-05-05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997.05.05');
mysql> INSERT INTO tbl_name (idate) VALUES ('1997 05 05');
```

```
mysql> INSERT INTO tbl_name (idate) VALUES ('0000-00-00');

mysql> SELECT idate FROM tbl_name WHERE idate >= '1997-05-05';
mysql> SELECT idate FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT mod(idate,100) FROM tbl_name WHERE idate >= 19970505;
mysql> SELECT idate FROM tbl_name WHERE idate >= '19970505';
```

However, the following will not work:

```
mysql> SELECT idate FROM tbl_name WHERE STRCMP(idate,'19970505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string and performs a string comparison. It does not convert `'19970505'` to a date and perform a date comparison.

Note that **MySQL** does no checking whether or not the date is correct. If you store an incorrect date, such as `'1998-2-31'`, the wrong date will be stored. If the date cannot be converted to any reasonable value, a `0` is stored in the `DATE` field. This is mainly a speed issue and we think it is up to the application to check the dates, and not the server.

18.13 Timezone problems

If you have a problem with `SELECT NOW()` returning values in GMT and not your local time, you have to set the `TZ` environment variable to your current timezone. This should be done for the environment in which the server runs, for example in `safe_mysqld` or `mysql.server`.

18.14 Case sensitivity in searches

By default, **MySQL** searches are case-insensitive (although there are some character sets that are never case insensitive, such as `czech`). That means that if you search with `col_name LIKE 'a%'`, you will get all column values that start with `A` or `a`. If you want to make this search case-sensitive, use something like `INDEX(col_name, "A")=0` to check a prefix. Or use `STRCMP(col_name, "A") = 0` if the column value must be exactly `"A"`.

Simple comparison operations (`>=, >, = , < , <=`, sorting and grouping) are based on each character's "sort value". Characters with the same sort value (like E, e and é) are treated as the same character!

`LIKE` comparisons are done on the uppercase value of each character (E == e but E <> é)

If you want a column always to be treated in case-sensitive fashion, declare it as `BINARY`. See Section 7.6 [CREATE TABLE], page 167.

If you are using Chinese data in the so-called big5 encoding, you want to make all character columns `BINARY`. This works because the sorting order of big5 encoding characters is based on the order of ASCII codes.

18.15 Problems with `NULL` values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string `''`. This is not the case! For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ("");
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as "phone number is not known" and the meaning of the second can be regarded as "she has no phone".

In SQL, the `NULL` value is always false in comparison to any other value, even `NULL`. An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return `NULL`:

```
mysql> SELECT NULL,1+NULL,CONCAT('Invisible',NULL);
```

If you want to search for column values that are `NULL`, you cannot use the `=NULL` test. The following statement returns no rows, because `expr = NULL` is FALSE, for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for `NULL` values, you must use the `IS NULL` test. The following shows how to find the `NULL` phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = "";
```

In **MySQL**, as in many other SQL servers, you can't index columns that can have `NULL` values. You must declare such columns `NOT NULL`. Conversely, you cannot insert `NULL` into an indexed column.

When reading data with `LOAD DATA INFILE`, empty columns are updated with `''`. If you want a `NULL` value in a column, you should use `\N` in the text file. The literal word `'NULL'` may also be used under some circumstances. See Section 7.15 [LOAD DATA], page 183.

When using `ORDER BY`, `NULL` values are presented first. If you sort in descending order using `DESC`, `NULL` values are presented last. When using `GROUP BY`, all `NULL` values are regarded as equal.

To help with `NULL` handling, you can use the `IS NULL` and `IS NOT NULL` operators and the `IFNULL()` function.

For some column types, `NULL` values are handled specially. If you insert `NULL` into the first `TIMESTAMP` column of a table, the current date and time is inserted. If you insert `NULL` into an `AUTO_INCREMENT` column, the next number in the sequence is inserted.

18.16 Problems with `alias`

You can use alias to refer to a column in the `GROUP BY`, `ORDER BY` or in the `HAVING` part. Aliases can also be used to give columns more better names:

```
SELECT SQRT(a*b) as rt FROM table_name GROUP BY rt HAVING rt > 0;
SELECT id,COUNT(*) AS cnt FROM table_name GROUP BY id HAVING cnt > 0;
SELECT id AS "Customer identity" FROM table_name;
```

Note that you ANSI SQL doesn't allow you to refer to an alias in a `WHERE` clause. This is because that when the `WHERE` code is executed the column value may not yet be determinated. For example the following query is **illegal**:

```
SELECT id,COUNT(*) AS cnt FROM table_name WHERE cnt > 0 GROUP BY id;
```

The `WHERE` statement is executed to determinate which rows should be included in the `GROUP BY` part while `HAVING` is used to decide which rows from the result set should be used.

18.17 Deleting rows from related tables

As **MySQL** doesn't support sub-selects or use of more than one table in the `DELETE` statement, you should use the following approach to delete rows from 2 related tables:

1. `SELECT` the rows based on some `WHERE` condition in the main table.
2. `DELETE` the rows in the main table based on the same condition.
3. `DELETE FROM related_table WHERE related_column IN (selected_rows)`

If the total number of characters in the query with `related_column` is more than 1,048,576 (the default value of `max_allowed_packet`, you should split it into smaller parts and execute multiple `DELETE` statements. You will probably get the fastest `DELETE` by only deleting 100-1000 `related_column` id's per time if the `related_column` is an index. If the `related_column` isn't an index, the speed is independent of the number of arguments in the `IN` clause.

18.18 Solving problems with no matching rows

If you have a complicated query with many tables that doesn't return any rows, you should use the following procedure to find out what is wrong with your query:

1. Test the query with `EXPLAIN` and check if you can find something that is obviously wrong. See Section 7.21 [EXPLAIN], page 194.
2. Select only those fields that are used in the `WHERE` clause.
3. Remove one table at a time from the query until it returns some rows. If the tables are big, it's a good idea to use `LIMIT 10` with the query.
4. Do a `SELECT` for the column that should have matched a row, against the table that was last removed from the query.
5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you can't use `=`! This problem is common in most computer languages because floating point values are not exact values.

   ```
   mysql> SELECT * FROM table_name WHERE float_column=3.5;
      ->
   mysql> SELECT * FROM table_name WHERE float_column between 3.45 and 3.55;
   ```

   In most cases, changing the `FLOAT` to a `DOUBLE` will fix this!
6. If you still can't find out what's wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file with `mysqldump --quick database tables > query.sql`. Take the file up in a editor, remove some insert lines (if there are too many of these) and add your select statement last in the file.

   Test that you still have your problem by doing:

```
shell> mysqladmin create test2
shell> mysql test2 < query.sql
```

Post the test file using `mysqlbug` to mysql@lists.mysql.com.

18.19 Problems with `ALTER TABLE`.

If `ALTER TABLE` dies with an error like this:

```
Error on rename of './database/name.frm' to './database/B-a.frm' (Errcode: 17)
```

The problem may be that **MySQL** has crashed in a previous `ALTER TABLE` and there is an old table named '`A-something`' or '`B-something`' lying around. In this case, go to the **MySQL** data directory and delete all files that have names starting with `A-` or `B-`. (You may want to move them elsewhere instead of deleting them).

`ALTER TABLE` works the following way:

- Create a new table named '`A-xxx`' with the requested changes.
- All rows from the old table are copied to '`A-xxx`'.
- The old table is renamed '`B-xxx`'.
- '`A-xxx`' is renamed to your old table name.
- '`B-xxx`' is deleted.

If something goes wrong with the renaming operation, **MySQL** tries to undo the changes. If something goes seriously wrong (this shouldn't happen, of course), **MySQL** may leave the old table as '`B-xxx`' but a simple rename should get your data back.

18.20 How to change the order of columns in a table

The whole point of SQL is to abstract the application from the data storage format. You should always specify the order in wish you wish to retrieve your data. For example:

```
SELECT col_name1, col_name2, col_name3 FROM tbl_name;
```

will return columns in the order `col_name1`, `col_name2`, `col_name3`, whereas:

```
SELECT col_name1, col_name3, col_name2 FROM tbl_name;
```

will return columns in the order `col_name1`, `col_name3`, `col_name2`.

You should **NEVER**, in an application, use `SELECT *` and retrieve the columns based on their position, because the order in which columns are returned **CANNOT** be guaranteed over time; A simple change to your database may cause your application to fail rather dramatically.

If you want to change the order of columns anyway, you can do it as follows:

1. Create a new table with the columns in the right order.
2. Execute `INSERT INTO new_table SELECT fields-in-new_table-order FROM old_table`.
3. Drop or rename `old_table`
4. `ALTER TABLE new_table RENAME old_table`

19 Solving some common problems with MySQL

19.1 Database replication

The most general way to replicate a database is to use the update log. See Section 9.2 [Update log], page 242. This requires one database that acts as a master (to which data changes are made) and one or more other databases that act as slaves. To update a slave, just run `mysql < update_log`. Supply host, user and password options that are appropriate for the slave database, and use the update log from the master database as input.

If you never delete anything from a table, you can use a `TIMESTAMP` column to find out which rows have been inserted or changed in the table since the last replication (by comparing to the time when you did the replication last time) and only copy these rows to the mirror.

It is possible to make a two-way updating system using both the update log (for deletes) and timestamps (on both sides). But in that case you must be able to handle conflicts when the same data have been changed in both ends. You probably want to keep the old version to help with deciding what has been updated.

Because replication in this case is done with SQL statements, you should not use the following functions in statements that update the database; they may not return the same value as in the original database:

- `DATABASE()`
- `GET_LOCK()` and `RELEASE_LOCK()`
- `RAND()`
- `USER()`, `SYSTEM_USER()` or `SESSION_USER()`
- `VERSION()`

All time functions are safe to use, as the timestamp is sent to the mirror if needed. `LAST_INSERT_ID()` is also safe to use.

19.2 Database backups

Since **MySQL** tables are stored as files, it is easy to do a backup. To get a consistent backup, do a `LOCK TABLES` on the relevant tables. See Section 7.23 [LOCK TABLES], page 198. You only need a read lock; this allows other threads to continue to query the tables while you are making a copy of the files in the database directory. If you want to make a SQL level backup, you can use `SELECT INTO OUTFILE`.

Another way to backup a database is to use the `mysqldump` program:

1. Do a full backup of your databases:

   ```
   shell> mysqldump --tab=/path/to/some/dir --lock-tables --opt
   ```

   You can also simply copy all table files (‘`*.frm`’, ‘`*.ISD`’ and ‘`*.ISM`’ files), as long as the server isn’t updating anything.
2. Stop `mysqld` if it’s running, then start it with the `--log-update` option. You will get log files with names of the form ‘*`hostname`*`.n`’, where `n` is a number that is incremented each time you execute `mysqladmin refresh` or `mysqladmin flush-logs`, the `FLUSH`

`LOGS` statement, or restart the server. These log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you executed `mysqldump`.

If you have to restore something, try to recover your tables using `isamchk -r` first. That should work in 99.9% of all cases. If `isamchk` fails, try the following procedure:

1. Restore the original `mysqldump` backup.
2. Execute the following command to re-run the updates in the update logs:

```
shell> ls -1 -t -r hostname.[0-9]* | xargs cat | mysql
```

`ls` is used to get all the log files in the right order.

You can also do selective backups with `SELECT * INTO OUTFILE 'file_name' FROM tbl_name` and restore with `LOAD DATA INFILE 'file_name' REPLACE ...` To avoid duplicate records, you need a `PRIMARY KEY` or a `UNIQUE` key in the table. The `REPLACE` keyword causes old records to be replaced with new ones when a new record duplicates an old record on a unique key value.

19.3 Running multiple MySQL servers on the same machine

There are circumstances when you might want to run multiple servers on the same machine. For example, you might want to test a new **MySQL** release while leaving your existing production setup undisturbed. Or you might be an Internet service provider that wants to provide independent **MySQL** installations for different customers.

If you want to run multiple servers, the easiest way is to compile the servers with different TCP/IP ports and socket files so they are not both listening to the same TCP/IP port or socket file.

Assume an existing server is configured for the default port number and socket file. Then configure the new server with a `configure` command something like this:

```
shell> ./configure  --with-tcp-port=port_number \
             --with-unix-socket=file_name \
             --prefix=/usr/local/mysql-3.22.9
```

Here `port_number` and `file_name` should be different than the default port number and socket file pathname, and the `--prefix` value should specify an installation directory different than the one under which the existing **MySQL** installation is located.

You can check the socket and port used by any currently-executing **MySQL** server with this command:

```
shell> mysqladmin -h hostname --port=port_number variables
```

If you have a **MySQL** server running on the port you used, you will get a list of some of the most important configurable variables in **MySQL**, including the socket name.

You should also edit the initialization script for your machine (probably '`mysql.server`') to start and kill multiple `mysqld` servers.

You don't have to recompile a new **MySQL** server just to start with a different port and socket. You can change the port and socket to be used by specifying them at runtime as options to `safe_mysqld`:

```
shell> /path/to/safe_mysqld --socket=file_name --port=port_number
```

If you run the new server on the same database directory as another server with logging enabled, you should also specify the name of the log files to `safe_mysqld` with `--log` and `--log-update`. Otherwise, both servers may be trying to write to the same log file.

Warning: Normally you should never have two servers that update data in the same database! If your OS doesn't support fault-free system locking, this may lead to unpleasant surprises!

If you want to use another database directory for the second server, you can use the `--datadir=path` option to `safe_mysqld`.

When you want to connect to a **MySQL** server that is running with a different port than the port that is compiled into your client, you can use one of the following methods:

- Start the client with `--host 'hostname' --port=port_numer` or `[--host localhost] --socket=file_name`.
- In your C or Perl programs, you can give the port and socket arguments when connecting to the **MySQL** server.
- Set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket and TCP/IP port before you start your clients. If you normally use a specific socket or port, you should place commands to set these environment variables in your '`.login`' file. See Section 12.1 [Programs], page 266.
- Specify the default socket and TCP/IP port in the '`.my.cnf`' file in your home directory. See Section 4.15.4 [Option files], page 79.

20 MySQL client tools and APIs

20.1 MySQL C API

The C API code is distributed with **MySQL**. It is included in the `mysqlclient` library and allows C programs to access a database.

Many of the clients in the MySQL source distribution are written in C. If you are looking for examples that demonstrate how to use the C API, take a look at these clients.

Most of the other client APIs (all except Java) use the `mysqlclient` library to communicate with the **MySQL** server. This means that, for example, you can take advantage of many of the same environment variables that are used by other client programs, because they are referenced from the library. See Section 12.1 [Programs], page 266, for a list of these variables.

The client has a maximum communication buffer size. The size of the buffer that is allocated initially (16K bytes) is automatically increased up to the maximum size (the default maximum is 24M). Since buffer sizes are increased only as demand warrants, simply increasing the default maximum limit does not in inself cause more resources to be used. This size check is mostly a check for erroneous queries and communication packets.

The communication buffer must be large enough to contain a single SQL statement (for client-to-server traffic) and one row of returned data (for server-to-client traffic). Each thread's communication buffer is dynamically enlarged to handle any query or row up to the maximum limit. For example, if you have `BLOB` values that contain up to 16M of data, you must have a communication buffer limit of at least 16M (in both server and client). The client's default maximum is 24M, but the default maximum in the server is 1M. You can increase this by changing the value of the `max_allowed_packet` parameter when the server is started. See Section 10.1 [Server parameters], page 244.

The **MySQL** server shrinks each communication buffer to `net_buffer_length` bytes after each query. For clients, the size of the buffer associated with a connection is not decreased until the connection is closed, at which time client memory is reclaimed.

20.2 C API datatypes

`MYSQL` This structure represents a handle to one database connection. It is used for almost all **MySQL** functions.

`MYSQL_RES`
This structure represents the result of a query that returns rows (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`). The information returned from a query is called the *result set* in the remainder of this section.

`MYSQL_ROW`
This is a type-safe representation of one row of data. It is currently implemented as an array of counted byte strings. (You cannot treat these as null-terminated strings if field values may contain binary data, because such values may contain null bytes internally.) Rows are obtained by calling `mysql_fetch_row()`.

`MYSQL_FIELD`
: This structure contains information about a field, such as the field's name, type and size. Its members are described in more detail below. You may obtain the `MYSQL_FIELD` structures for each field by calling `mysql_fetch_field()` repeatedly. Field values are not part of this structure; they are contained in a `MYSQL_ROW` structure.

`MYSQL_FIELD_OFFSET`
: This is a type-safe representation of an offset into a **MySQL** field list. (Used by `mysql_field_seek()`.) Offsets are field numbers within a row, beginning at zero.

`my_ulonglong`
: The type used for the number of rows and for `mysql_affected_rows()`, `mysql_num_rows()` and `mysql_insert_id()`. This type provides a range of `0` to `1.84e19`.

  On some systems, attempting to print a value of type `my_ulonglong` will not work. To print such a value, convert it to `unsigned long` and use a `%lu` print format. Example:

```
printf (Number of rows: %lu\n", (unsigned long) mysql_num_rows(result));
```

The `MYSQL_FIELD` structure contains the members listed below:

`char * name`
: The name of the field.

`char * table`
: The name of the table containing this field, if it isn't a calculated field. For calculated fields, the `table` value is a `NULL` pointer.

`char * def`
: The default value of this field (set only if you use `mysql_list_fields()`).

`enum enum_field_types type`
: The type of the field. The `type` value may be one of the following:

Type value	**Type meaning**
`FIELD_TYPE_TINY`	`TINYINT` field
`FIELD_TYPE_SHORT`	`SMALLINT` field
`FIELD_TYPE_LONG`	`INTEGER` field
`FIELD_TYPE_INT24`	`MEDIUMINT` field
`FIELD_TYPE_LONGLONG`	`BIGINT` field
`FIELD_TYPE_DECIMAL`	`DECIMAL` or `NUMERIC` field
`FIELD_TYPE_FLOAT`	`FLOAT` field
`FIELD_TYPE_DOUBLE`	`DOUBLE` or `REAL` field
`FIELD_TYPE_TIMESTAMP`	`TIMESTAMP` field
`FIELD_TYPE_DATE`	`DATE` field
`FIELD_TYPE_TIME`	`TIME` field
`FIELD_TYPE_DATETIME`	`DATETIME` field
`FIELD_TYPE_YEAR`	`YEAR` field

FIELD_TYPE_STRING	String (CHAR or VARCHAR) field
FIELD_TYPE_BLOB	BLOB or TEXT field (use max_length to determine the maximum length)
FIELD_TYPE_SET	SET field
FIELD_TYPE_ENUM	ENUM field
FIELD_TYPE_NULL	NULL-type field
FIELD_TYPE_CHAR	Deprecated; use FIELD_TYPE_TINY instead

You can use the IS_NUM() macro to test whether or not a field has a numeric type. Pass the type value to IS_NUM() and it will evaluate to TRUE if the field is numeric:

```
if (IS_NUM(field->type))
    printf("Field is numeric\n");
```

unsigned int length
: The width of the field.

unsigned int max_length
: The maximum width of the field for the result set. If you used mysql_list_fields(), this contains the maximum length for the field.

unsigned int flags
: Different bit-flags for the field. The flags value may have zero or more of the following bits set:

Flag value	Flag meaning
NOT_NULL_FLAG	Field can't be NULL
PRI_KEY_FLAG	Field is part of a primary key
UNIQUE_KEY_FLAG	Field is part of a unique key
MULTIPLE_KEY_FLAG	Field is part of a non-unique key.
UNSIGNED_FLAG	Field has the UNSIGNED attribute
ZEROFILL_FLAG	Field has the ZEROFILL attribute
BINARY_FLAG	Field has the BINARY attribute
AUTO_INCREMENT_FLAG	Field has the AUTO_INCREMENT attribute
ENUM_FLAG	Field is an ENUM (deprecated)
BLOB_FLAG	Field is a BLOB or TEXT (deprecated)
TIMESTAMP_FLAG	Field is a TIMESTAMP (deprecated)

Use of the BLOB_FLAG, ENUM_FLAG and TIMESTAMP_FLAG flags is deprecated because they indicate the type of a field rather than an attribute of its type. It is preferable to test field->type against FIELD_TYPE_BLOB, FIELD_TYPE_ENUM or FIELD_TYPE_TIMESTAMP instead.

The example below illustrates a typical use of the flags value:

```
if (field->flags & NOT_NULL_FLAG)
    printf("Field can't be null\n");
```

You may use the following convenience macros to determine the boolean status of the flags value:

IS_NOT_NULL(flags)	True if this field is defined as NOT NULL
IS_PRI_KEY(flags)	True if this field is a primary key

`IS_BLOB(flags)` — True if this field is a `BLOB` or `TEXT` (deprecated; test `field->type` instead)

`unsigned int decimals`
The number of decimals for numeric fields.

20.3 C API function overview

The functions available in the C API are listed below and are described in greater detail in the next section. See Section 20.4 [C API functions], page 330.

mysql_affected_rows() — Returns the number of rows affected by the last `UPDATE`, `DELETE` or `INSERT` query.

mysql_close() — Closes a server connection.

mysql_connect() — Connects to a **MySQL** server. This function is deprecated; use `mysql_real_connect()` instead.

mysql_change_user() — Change user and database on an open connection.

mysql_create_db() — Creates a database. This function is deprecated; use the SQL command `CREATE DATABASE` instead.

mysql_data_seek() — Seeks to an arbitrary row in a query result set.

mysql_debug() — Does a `DBUG_PUSH` with the given string.

mysql_drop_db() — Drops a database. This function is deprecated; use the SQL command `DROP DATABASE` instead.

mysql_dump_debug_info() — Makes the server write debug information to the log.

mysql_eof() — Determines whether or not the last row of a result set has been read. This function is deprecated; `mysql_errno()` or `mysql_error()` may be used instead.

mysql_errno() — Returns the error number for the most recently invoked **MySQL** function.

mysql_error() — Returns the error message for the most recently invoked **MySQL** function.

mysql_escape_string() — Escapes special characters in a string for use in a SQL statement.

mysql_fetch_field() — Returns the type of the next table field.

mysql_fetch_field_direct() — Returns the type of a table field, given a field number.

mysql_fetch_fields() — Returns an array of all field structures.

mysql_fetch_lengths() — Returns the lengths of all columns in the current row.

mysql_fetch_row() — Fetches the next row from the result set.

mysql_field_seek()	Puts the column cursor on a specified column.
mysql_field_count()	Returns the number of result columns for the most recent query.
mysql_field_tell()	Returns the position of the field cursor used for the last `mysql_fetch_field()`.
mysql_free_result()	Frees memory used by a result set.
mysql_get_client_info()	Returns client version information.
mysql_get_host_info()	Returns a string describing the connection.
mysql_get_proto_info()	Returns the protocol version used by the connection.
mysql_get_server_info()	Returns the server version number.
mysql_info()	Returns information about the most recently executed query.
mysql_init()	Gets or initializes a `MYSQL` structure.
mysql_insert_id()	Returns the ID generated for an `AUTO_INCREMENT` column by the previous query.
mysql_kill()	Kill a given thread.
mysql_list_dbs()	Returns database names matching a simple regular expression.
mysql_list_fields()	Returns field names matching a simple regular expression.
mysql_list_processes()	Returns a list of the current server threads.
mysql_list_tables()	Returns table names matching a simple regular expression.
mysql_num_fields()	Returns the number of columns in a result set.
mysql_num_rows()	Returns the number of rows in a result set.
mysql_options()	Set connect options for `mysql_connect()`.
mysql_ping()	Checks whether or not the connection to the server is working, reconnecting as necessary.
mysql_query()	Executes a SQL query specified as a null-terminated string.
mysql_real_connect()	Connects to a **MySQL** server.
mysql_real_query()	Executes a SQL query specified as a counted string.
mysql_reload()	Tells the server to reload the grant tables.
mysql_row_seek()	Seeks to a row in a result set, using value returned from `mysql_row_tell()`.

mysql_row_tell()	Returns the row cursor position.
mysql_select_db()	Connects to a database.
mysql_shutdown()	Shuts down the database server.
mysql_stat()	Returns the server status as a string.
mysql_store_result()	Retrieves a complete result set to the client.
mysql_thread_id()	Returns the current thread ID.
mysql_use_result()	Initiates a row-by-row result set retrieval.

To connect to the server, call `mysql_init()` to initialize a connection handler, then call `mysql_real_connect()` with that handler (along with other information such as the hostname, user name and password). When you are done with the connection, call `mysql_close()` to terminate it.

While a connection is active, the client may send SQL queries to the server using `mysql_query()` or `mysql_real_query()`. The difference between the two is that `mysql_query()` expects the query to be specified as a null-terminated string whereas `mysql_real_query()` expects a counted string. If the string contains binary data (which may include null bytes), you must use `mysql_real_query()`.

For each non-`SELECT` query (e.g., `INSERT`, `UPDATE`, `DELETE`), you can found out how many rows were affected (changed) by calling `mysql_affected_rows()`.

For `SELECT` queries, you retrieve the selected rows as a result set. (Note that some statements are `SELECT`-like in that they return rows. These include `SHOW`, `DESCRIBE` and `EXPLAIN`. They should be treated the same way as `SELECT` statements.)

There are two ways for a client to process result sets. One way is to retrieve the entire result set all at once by calling `mysql_store_result()`. This function acquires from the server all the rows returned by the query and stores them in the client. The second way is for the client to initiate a row-by-row result set retrieval by calling `mysql_use_result()`. This function initializes the retrieval, but does not actually get any rows from the server.

In both cases, you access rows by calling `mysql_fetch_row()`. With `mysql_store_result()`, `mysql_fetch_row()` accesses rows that have already been fetched from the server. With `mysql_use_result()`, `mysql_fetch_row()` actually retrieves the row from the server. Information about as the size of the data values in each row is available by calling `mysql_fetch_lengths()`.

After you are done with a result set, call `mysql_free_result()` to free the memory used for it.

The two retrieval mechanisms are complementary. Client programs should choose the approach that is most appropriate for their requirements. In practice, clients tend to use `mysql_store_result()` more commonly.

An advantage of `mysql_store_result()` is that since the rows have all been fetched to the client, you not only can access rows sequentially, you can move back and forth in the result set using `mysql_data_seek()` or `mysql_row_seek()` to change the current row position within the result set. You can also find out how many rows there are by calling `mysql_num_rows()`. On the other hand, the memory requirements for `mysql_store_result()`

may be very high for large result sets and you are more likely to encounter out-of-memory conditions.

An advantage of `mysql_use_result()` is that the client requires less memory for the result set since it maintains only one row at a time (and since there is less allocation overhead, `mysql_use_result()` can be faster). Disadvantages are that you must process each row quickly to avoid tying up the server, you don't have random access to rows within the result set (you can only access rows sequentially), and you don't know how many rows are in the result set until you have retrieved them all. Furthermore, you *must* retrieve all the rows even if you determine in mid-retrieval that you've found the information you were looking for.

The API makes it possible for clients to respond appropriately to queries (retrieving rows only as necessary) without knowing whether or not the query is a `SELECT`. You can do this by calling `mysql_store_result()` after each `mysql_query()` (or `mysql_real_query()`). If the result set call succeeds, the query was a `SELECT` and you can read the rows. If the result set call fails, call `mysql_field_count()` to determine whether or not a result was actually to be expected. If `mysql_field_count()` returns zero, the query returned no data (indicating that it was an `INSERT`, `UPDATE`, `DELETE`, etc.), and thus not expected to return rows. If `mysql_field_count()` is non-zero, the query should have returned rows, but didn't. This indicates that the query was a `SELECT` that failed. See the description for `mysql_field_count()` for an example of how this can be done.

Both `mysql_store_result()` and `mysql_use_result()` allow you to obtain information about the fields that make up the result set (the number of fields, their names and types, etc.). You can access field information sequentially within the row by calling `mysql_fetch_field()` repeatedly, or by field number within the row by calling `mysql_fetch_field_direct()`. The current field cursor position may be changed by calling `mysql_field_seek()`. Setting the field cursor affects subsequent calls to `mysql_fetch_field()`. You can also get information for fields all at once by calling `mysql_fetch_fields()`.

For detecting and reporting errors, **MySQL** provides access to error information by means of the `mysql_errno()` and `mysql_error()` functions. These return the error code or error message for the most recently invoked function that can succeed or fail, allowing you to determine when an error occurred and what it was.

20.4 C API function descriptions

In the descriptions below, a parameter or return value of `NULL` means `NULL` in the sense of the C programming language, not a **MySQL** `NULL` value.

Functions that return a value generally return a pointer or an integer. Unless specified otherwise, functions returning a pointer return a non-`NULL` value to indicate success or a `NULL` value to indicate an error, and functions returning an integer return zero to indicate success or non-zero to indicate an error. Note that "non-zero" means just that. Unless the function description says otherwise, do not test against a value other than zero:

```
if (result)                     /* correct */
    ... error ...

if (result < 0)                 /* incorrect */
```

```
    ... error ...

if (result == -1)             /* incorrect */
    ... error ...
```

When a function returns an error, the **Errors** subsection of the function description lists the possible types of errors. You can find out which of these occurred by calling `mysql_errno()`. A string representation of the error may be obtained by calling `mysql_error()`.

20.4.1 `mysql_affected_rows()`

`my_ulonglong mysql_affected_rows(MYSQL *mysql)`

Description

Returns the number of rows affected (changed) by the last `UPDATE`, `DELETE` or `INSERT` query. May be called immediately after `mysql_query()` for `UPDATE`, `DELETE` or `INSERT` statements. For `SELECT` statements, `mysql_affected_rows()` works like `mysql_num_rows()`.

`mysql_affected_rows()` is currently implemented as a macro.

Return values

An integer greater than zero indicates the number of rows affected or retrieved. Zero indicates that no records matched the `WHERE` clause in the query or that no query has yet been executed. -1 indicates that the query returned an error or that, for a `SELECT` query, `mysql_affected_rows()` was called prior to calling `mysql_store_result()`.

Errors

None.

Example

```
mysql_query(&mysql,"UPDATE products SET cost=cost*1.25 WHERE group=10");
printf("%d products updated",mysql_affected_rows(&mysql));
```

20.4.2 `mysql_close()`

`void mysql_close(MYSQL *mysql)`

Description

Closes a previously opened connection. `mysql_close()` also deallocates the connection handle pointed to by `mysql` if the handle was allocated automatically by `mysql_init()` or `mysql_real_connect()`.

Return values

None.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.3 `mysql_connect()`

```
MYSQL *mysql_connect(MYSQL *mysql, const char *host, const char *user, const
char *passwd)
```

Description

This function is deprecated. It is preferable to use `mysql_real_connect()` instead.

`mysql_connect()` attempts to establish a connection to a **MySQL** database engine running on `host`. `mysql_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The meanings of the parameters are the same as for the corresponding parameters for `mysql_real_connect()`. See the description of that function for more information.

Return values

Same as for `mysql_real_connect()`.

Errors

Same as for `mysql_real_connect()`.

20.4.4 `mysql_change_user()`

```
my_bool mysql_change_user(MYSQL *mysql, const char *user, const char
*password, const char *db)
```

Description

Changes the user and causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

This function was introduced in **MySQL** 3.23.3

`mysql_cohange_user()` fails unless the connected user can be authenticated or if he doesn't have permission to use the database. In this case the user and database is not changed

Return values

Zero for success. Non-zero if an error occurred.

Errors

The same that you can get from `mysql_real_connect()`

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

`ER_UNKNOWN_COM_ERROR`
: The **MySQL** server doesn't implement this command (probably an old server)

`ER_ACCESS_DENIED_ERROR`
: The user or password was wrong.

`ER_BAD_DB_ERROR`
: The database didn't exists.

`ER_DBACCESS_DENIED_ERROR`
: The user did not have access rights to the database.

`ER_WRONG_DB_NAME`
: The database name was too long.

Example

```
if (mysql_change_user(&mysql, "user", "password", "new_database"))
{
   fprintf(stderr, "Failed to change user.  Error: %s\n",
           mysql_error(&mysql));
}
```

20.4.5 `mysql_create_db()`

```
int mysql_create_db(MYSQL *mysql, const char *db)
```

Description

Creates the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue a SQL `CREATE DATABASE` statement instead.

Return values

Zero if the database was created successfully. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

Example

```
if(mysql_create_db(&mysql, "my_database"))
{
   fprintf(stderr, "Failed to create new database.  Error: %s\n",
           mysql_error(&mysql));
}
```

20.4.6 `mysql_data_seek()`

`void mysql_data_seek(MYSQL_RES *result, unsigned int offset)`

Description

Seeks to an arbitrary row in a query result set. This requires that the result set structure contains the entire result of the query, so `mysql_data_seek()` may be used in conjunction only with `mysql_store_result()`, not with `mysql_use_result()`.

The offset should be a value in the range from 0 to `mysql_num_rows(result)-1`.

Return values

None.

Errors

None.

20.4.7 `mysql_debug()`

`void mysql_debug(char *debug)`

Description

Does a `DBUG_PUSH` with the given string. `mysql_debug()` uses the Fred Fish debug library. To use this function, you must compile the client library to support debugging. See Section G.1 [Debugging server], page 444. See Section G.2 [Debugging client], page 446.

Return values

None.

Errors

None.

Example

The call shown below causes the client library to generate a trace file in '`/tmp/client.trace`' on the client machine:

```
mysql_debug("d:t:O,/tmp/client.trace");
```

20.4.8 `mysql_drop_db()`

```
int mysql_drop_db(MYSQL *mysql, const char *db)
```

Description

Drops the database named by the `db` parameter.

This function is deprecated. It is preferable to use `mysql_query()` to issue a SQL `DROP DATABASE` statement instead.

Return values

Zero if the database was dropped successfully. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

Example

```
if(mysql_drop_db(&mysql, "my_database"))
  fprintf(stderr, "Failed to drop the database: Error: %s\n",
          mysql_error(&mysql));
```

20.4.9 mysql_dump_debug_info()

```
int mysql_dump_debug_info(MYSQL *mysql)
```

Description

Instructs the server to write some debug information to the log. The connected user must have the **process** privilege for this to work.

Return values

Zero if the command was successful. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.10 mysql_eof()

```
my_bool mysql_eof(MYSQL_RES *result)
```

Description

This function is deprecated. `mysql_errno()` or `mysql_error()` may be used instead.

`mysql_eof()` determines whether or not the last row of a result set has been read.

If you acquire a result set from a successful call to `mysql_store_result()`, the client receives the entire set in one operation. In this case, a `NULL` return from `mysql_fetch_row()` always means the end of the result set has been reached and it is unnecessary to call `mysql_eof()`.

On the other hand, if you use `mysql_use_result()` to initiate a result set retrieval, the rows of the set are obtained from the server one by one as you call `mysql_fetch_row()` repeatedly. Because an error may occur on the connection during this process, a `NULL`

return value from `mysql_fetch_row()` does not necessarily mean the end of the result set was reached normally. In this case, you can use `mysql_eof()` to determine what happened. `mysql_eof()` returns a non-zero value if the end of the result set was reached and zero if an error occurred.

Historically, `mysql_eof()` predates the standard **MySQL** error functions `mysql_errno()` and `mysql_error()`. Since those error functions provide the same information, their use is preferred over `mysql_eof()`, which is now deprecated. (In fact, they provide more information, since `mysql_eof()` returns only a boolean value whereas the error functions indicate a reason for the error when one occurs.)

Return values

Zero if an error occurred. Non-zero if the end of the result set has been reached.

Errors

None.

Example

The following example shows how you might use `mysql_eof()`:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(!mysql_eof(result))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

However, you can achieve the same effect with the standard **MySQL** error functions:

```
mysql_query(&mysql,"SELECT * FROM some_table");
result = mysql_use_result(&mysql);
while((row = mysql_fetch_row(result)))
{
    // do something with data
}
if(mysql_errno(&mysql))  // mysql_fetch_row() failed due to an error
{
    fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
```

20.4.11 mysql_errno()

```
unsigned int mysql_errno(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_errno()` returns the error code for the most recently invoked API function that can succeed or fail. A return value of zero means that no error occurred. Client error message numbers are listed in the **MySQL** '`errmsg.h`' header file. Server error message numbers are listed in '`mysqld_error.h`'

Return values:

An error code value. Zero if no error occurred.

Errors

None.

20.4.12 `mysql_error()`

```
char *mysql_error(MYSQL *mysql)
```

Description

For the connection specified by `mysql`, `mysql_error()` returns the error message for the most recently invoked API function that can succeed or fail. An empty string (`""`) is returned if no error occurred. This means the following two tests are equivalent:

```
if(mysql_errno(&mysql))
{
    // an error occurred
}

if(mysql_error(&mysql)[0] != '\0')
{
    // an error occurred
}
```

The language of the client error messages may be changed by recompiling the **MySQL** client library. Currently you can choose error messages in several different languages. See Section 9.1 [Languages], page 240.

Return values

A character string that describes the error. An empty string if no error occurred.

Errors

None.

20.4.13 `mysql_escape_string()`

```
unsigned int mysql_escape_string(char *to, const char *from, unsigned int
length)
```

Description

Encodes the string in `from` to an escaped SQL string that can be sent to the server in a SQL statement, and places the result in `to`. Characters encoded are `NUL` (ASCII 0), '`\n`', '`\r`', '`\`' and '`'`' (see Section 7.1 [Literals], page 116).

The string pointed to by `from` must be `length` bytes long (not including the terminating null byte). You must allocate the `to` buffer to be at least `length*2+1` bytes long. When `mysql_escape_string()` returns, the contents of `to` will be a null-terminated string. The return value is the length of the encoded string, not including the terminating null character.

Example

```
char query[1000],*end;

end = strmov(query,"INSERT INTO test_table values(");
*end++ = '\'';
end += mysql_escape_string(end,"What's this",11);
*end++ = '\'';
*end++ = ',';
*end++ = '\'';
end += mysql_escape_string(end,"binary data: \0\r\n",16);
*end++ = '\'';
*end++ = ')';

if (mysql_real_query(&mysql,query,(unsigned int) (end - query)))
{
   fprintf(stderr, "Failed to insert row, Error: %s\n",
           mysql_error(&mysql));
}
```

The `strmov()` function used in the example is included in the `mysqlclient` library and works like `strcpy()` but returns a pointer to the terminating null of the first parameter.

Return values

The length of the value placed into `to`, not including the terminating null character.

Errors

None.

20.4.14 `mysql_fetch_field()`

```
MYSQL_FIELD *mysql_fetch_field(MYSQL_RES *result)
```

Description

Returns the definition of one column of a result set as a `MYSQL_FIELD` structure. Call this function repeatedly to retrieve information about all columns in the result set. `mysql_fetch_field()` returns `NULL` when no more fields are left.

`mysql_fetch_field()` is reset to return information about the first field each time you execute a new `SELECT` query. The field returned by `mysql_fetch_field()` is also affected by calls to `mysql_field_seek()`.

If you've called `mysql_query()` to perform a `SELECT` on a table but have not called `mysql_store_result()`, **MySQL** returns the default blob length (8K bytes) if you call `mysql_fetch_field()` to ask for the length of a `BLOB` field. (The 8K size is chosen because **MySQL** doesn't know the maximum length for the `BLOB`. This should be made configurable sometime.) Once you've retrieved the result set, `field->max_length` contains the length of the largest value for this column in the specific query.

Return values

The `MYSQL_FIELD` structure for the current column. `NULL` if no columns are left.

Errors

None.

Example

```
MYSQL_FIELD *field;

while((field = mysql_fetch_field(result)))
{
    printf("field name %s\n", field->name);
}
```

20.4.15 mysql_fetch_fields()

```
MYSQL_FIELD *mysql_fetch_fields(MYSQL_RES *result)
```

Description

Returns an array of all `MYSQL_FIELD` structures for a result set. Each structure provides the field definition for one column of the result set.

Return values

An array of `MYSQL_FIELD` structures for all columns of a result set.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *fields;

num_fields = mysql_num_fields(result);
fields = mysql_fetch_fields(result);
for(i = 0; i < num_fields; i++)
{
   printf("Field %u is %s\n", i, fields[i].name);
}
```

20.4.16 mysql_fetch_field_direct()

```
MYSQL_FIELD *mysql_fetch_field_direct(MYSQL_RES *result, unsigned int
fieldnr)
```

Description

Given a field number `fieldnr` for a column within a result set, returns that column's field definition as a `MYSQL_FIELD` structure. You may use this function to retrieve the definition for an arbitrary column. The value of `fieldnr` should be in the range from 0 to `mysql_num_fields(result)-1`.

Return values

The `MYSQL_FIELD` structure for the specified column.

Errors

None.

Example

```
unsigned int num_fields;
unsigned int i;
MYSQL_FIELD *field;

num_fields = mysql_num_fields(result);
for(i = 0; i < num_fields; i++)
{
    field = mysql_fetch_field_direct(result, i);
    printf("Field %u is %s\n", i, field->name);
}
```

20.4.17 mysql_fetch_lengths()

```
unsigned long *mysql_fetch_lengths(MYSQL_RES *result)
```

Description

Returns the lengths of the columns of the current row within a result set. If you plan to copy field values, this length information is also useful for optimization, because you can avoid calling `strlen()`. In addition, if the result set contains binary data, you *must* use this function to determine the size of the data, because `strlen()` returns incorrect results for any field containing null characters.

The length for empty columns and for columns containing `NULL` values is zero. To see how to distinguish these two cases, see the description for `mysql_fetch_row()`.

Return values

An array of unsigned long integers representing the size of each column (not including any terminating null characters). `NULL` if an error occurred.

Errors

`mysql_fetch_lengths()` is valid only for the current row of the result set. It returns `NULL` if you call it before calling `mysql_fetch_row()` or after retrieving all rows in the result.

Example

```
MYSQL_ROW row;
unsigned long *lengths;
unsigned int num_fields;
unsigned int i;

row = mysql_fetch_row(result);
if (row)
{
    num_fields = mysql_num_fields(result);
    lengths = mysql_fetch_lengths(result);
    for(i = 0; i < num_fields; i++)
    {
         printf("Column %u is %lu bytes in length.\n", i, lengths[i]);
    }
}
```

20.4.18 `mysql_fetch_row()`

`MYSQL_ROW mysql_fetch_row(MYSQL_RES *result)`

Description

Retrieves the next row of a result set. When used after `mysql_store_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve. When used after `mysql_use_result()`, `mysql_fetch_row()` returns `NULL` when there are no more rows to retrieve or if an error occurred.

The number of values in the row is given by `mysql_num_fields(result)`. If `row` holds the return value from a call to `mysql_fetch_row()`, pointers to the values are accessed as `row[0]` to `row[mysql_num_fields(result)-1]`. `NULL` values in the row are indicated by `NULL` pointers.

The lengths of the field values in the row may be obtained by calling `mysql_fetch_lengths()`. Empty fields and fields containing `NULL` both have length 0; you can distinguish these by checking the pointer for the field value. If the pointer is `NULL`, the field is `NULL`; otherwise the field is empty.

Return values

A `MYSQL_ROW` structure for the next row. `NULL` if there are no more rows to retrieve or if an error occurred.

Errors

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

Example

```
MYSQL_ROW row;
unsigned int num_fields;
unsigned int i;

num_fields = mysql_num_fields(result);
while ((row = mysql_fetch_row(result)))
{
   unsigned long *lengths;
   lengths = mysql_fetch_lengths(result);
   for(i = 0; i < num_fields; i++)
   {
       printf("[%.*s] ", (int) lengths[i], row[i] ? row[i] : "NULL");
   }
   printf("\n");
}
```

20.4.19 `mysql_field_count()`

`unsigned int mysql_field_count(MYSQL *mysql)`

If you are using a version of **MySQL** earlier than 3.22.24, you should use `unsigned int mysql_num_fields(MYSQL *mysql)` instead.

Description

Returns the number of columns for the most recent query on the connection.

The normal use of this function is when `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether or not `mysql_store_result()` should have produced a non-empty result. This allows the client program to take proper action without knowing whether or not the query was a `SELECT` (or `SELECT`-like) statement. The example shown below illustrates how this may be done.

See Section 20.4.51 [`NULL mysql_store_result()`], page 365.

Return values

An unsigned integer representing the number of fields in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;

if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if(mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
        else // mysql_store_result() should have returned data
        {
            fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
```

```
            }
        }
    }
```

An alternative is to replace the `mysql_field_count(&mysql)` call with `mysql_errno(&mysql)`. In this case, you are checking directly for an error from `mysql_store_result()` rather than inferring from the value of `mysql_field_count()` whether or not the statement was a `SELECT`.

20.4.20 `mysql_field_seek()`

```
MYSQL_FIELD_OFFSET mysql_field_seek(MYSQL_RES *result, MYSQL_FIELD_OFFSET
offset)
```

Description

Sets the field cursor to the given offset. The next call to `mysql_fetch_field()` will retrieve the field definition of the column associated with that offset.

To seek to the beginning of a row, pass an `offset` value of zero.

Return values

The previous value of the field cursor.

Errors

None.

20.4.21 `mysql_field_tell()`

```
MYSQL_FIELD_OFFSET mysql_field_tell(MYSQL_RES *result)
```

Description

Returns the position of the field cursor used for the last `mysql_fetch_field()`. This value can be used as an argument to `mysql_field_seek()`.

Return values

The current offset of the field cursor.

Errors

None.

20.4.22 `mysql_free_result()`

```
void mysql_free_result(MYSQL_RES *result)
```

Description

Frees the memory allocated for a result set by `mysql_store_result()`, `mysql_use_result()`, `mysql_list_dbs()`, etc. When you are done with a result set, you must free the memory it uses by calling `mysql_free_result()`.

Return values

None.

Errors

None.

20.4.23 `mysql_get_client_info()`

```
char *mysql_get_client_info(void)
```

Description

Returns a string that represents the client library version.

Return values

A character string that represents the **MySQL** client library version.

Errors

None.

20.4.24 `mysql_get_host_info()`

```
char *mysql_get_host_info(MYSQL *mysql)
```

Description

Returns a string describing the type of connection in use, including the server host name.

Return values

A character string representing the server host name and the connection type.

Errors

None.

20.4.25 `mysql_get_proto_info()`

```
unsigned int mysql_get_proto_info(MYSQL *mysql)
```

Description

Returns the protocol version used by current connection.

Return values

An unsigned integer representing the protocol version used by the current connection.

Errors

None.

20.4.26 mysql_get_server_info()

```
char *mysql_get_server_info(MYSQL *mysql)
```

Description

Returns a string that represents the server version number.

Return values

A character string that represents the server version number.

Errors

None.

20.4.27 mysql_info()

```
char *mysql_info(MYSQL *mysql)
```

Description

Retrieves a string providing information about the most recently executed query, but only for the statements listed below. For other statements, `mysql_info()` returns `NULL`. The format of the string varies depending on the type of query, as described below. The numbers are illustrative only; the string will contain values appropriate for the query.

`INSERT INTO ... SELECT ...`
: String format: `Records: 100 Duplicates: 0 Warnings: 0`

`INSERT INTO ... VALUES (...),(...),(...)...`
: String format: `Records: 3 Duplicates: 0 Warnings: 0`

`LOAD DATA INFILE ...`
: String format: `Records: 1 Deleted: 0 Skipped: 0 Warnings: 0`

`ALTER TABLE`
: String format: `Records: 3 Duplicates: 0 Warnings: 0`

Note that `mysql_info()` returns a non-`NULL` value for the `INSERT ... VALUES` statement only if multiple value lists are specified in the statement.

Return values

A character string representing additional information about the most recently executed query. `NULL` if no information is available for the query.

Errors

None.

20.4.28 `mysql_init()`

```
MYSQL *mysql_init(MYSQL *mysql)
```

Description

Allocates or initializes a `MYSQL` object suitable for `mysql_real_connect()`. If `mysql` is a `NULL` pointer, the function allocates, initializes and returns a new object. Otherwise the object is initialized and the address of the object is returned. If `mysql_init()` allocates a new object, it will be freed when `mysql_close()` is called to close the connection.

Return values

An initialized `MYSQL*` handle. `NULL` if there was insufficient memory to allocate a new object.

Errors

In case of insufficient memory, `NULL` is returned.

20.4.29 `mysql_insert_id()`

```
my_ulonglong mysql_insert_id(MYSQL *mysql)
```

Description

Returns the ID generated for an `AUTO_INCREMENT` column by the previous query. Use this function after you have performed an `INSERT` query into a table that contains an `AUTO_INCREMENT` field.

Note that `mysql_insert_id()` returns `0` if the previous query does not generate an `AUTO_INCREMENT` value. If you need to save the value for later, be sure to call `mysql_insert_id()` immediately after the query that generates the value.

Also note that the value of the SQL `LAST_INSERT_ID()` function always contains the most recently generated `AUTO_INCREMENT` value, and is not reset between queries since the value of that function is maintained in the server.

Return values

The value of the `AUTO_INCREMENT` field that was updated by the previous query. Returns zero if there was no previous query on the connection or if the query did not update an `AUTO_INCREMENT` value.

Errors

None.

20.4.30 `mysql_kill()`

```
int mysql_kill(MYSQL *mysql, unsigned long pid)
```

Description

Asks the server to kill the thread specified by `pid`.

Return values

Zero for success. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.31 `mysql_list_dbs()`

```
MYSQL_RES *mysql_list_dbs(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of database names on the server that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters '`%`' or '`_`', or may be a `NULL` pointer to match all databases. Calling `mysql_list_dbs()` is similar to executing the query `SHOW databases [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return values

A MYSQL_RES result set for success. NULL if an error occurred.

Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_OUT_OF_MEMORY
: Out of memory.

CR_SERVER_GONE_ERROR
: The **MySQL** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

CR_UNKNOWN_ERROR
: An unknown error occurred.

20.4.32 mysql_list_fields()

```
MYSQL_RES *mysql_list_fields(MYSQL *mysql, const char *table, const char
*wild)
```

Description

Returns a result set consisting of field names in the given table that match the simple regular expression specified by the wild parameter. wild may contain the wildcard characters '%' or '_', or may be a NULL pointer to match all fields. Calling mysql_list_fields() is similar to executing the query SHOW COLUMNS FROM tbl_name [LIKE wild].

Note that it's recommended that you use SHOW COLUMNS FROM tbl_name instead of mysql_list_fields().

You must free the result set with mysql_free_result().

Return values

A MYSQL_RES result set for success. NULL if an error occurred.

Errors

CR_COMMANDS_OUT_OF_SYNC
: Commands were executed in an improper order.

CR_SERVER_GONE_ERROR
: The **MySQL** server has gone away.

CR_SERVER_LOST
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.33 `mysql_list_processes()`

```
MYSQL_RES *mysql_list_processes(MYSQL *mysql)
```

Description

Returns a result set describing the current server threads. This is the same kind of information as that reported by `mysqladmin processlist`.

You must free the result set with `mysql_free_result()`.

Return values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.34 `mysql_list_tables()`

```
MYSQL_RES *mysql_list_tables(MYSQL *mysql, const char *wild)
```

Description

Returns a result set consisting of table names in the current database that match the simple regular expression specified by the `wild` parameter. `wild` may contain the wildcard characters '`%`' or '`_`', or may be a `NULL` pointer to match all tables. Calling `mysql_list_tables()` is similar to executing the query `SHOW tables [LIKE wild]`.

You must free the result set with `mysql_free_result()`.

Return values

A `MYSQL_RES` result set for success. `NULL` if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.35 `mysql_num_fields()`

```
unsigned int mysql_num_fields(MYSQL_RES *result)
```

or

```
unsigned int mysql_num_fields(MYSQL *mysql)
```

The second form doesn't work on **MySQL** 3.22.24 or newer. To pass a `MYSQL*` argument, you must use `unsigned int mysql_field_count(MYSQL *mysql)` instead.

Description

Returns the number of columns in a result set.

Note that you can get the number of columns either from a pointer to a result set or to a connection handle. You would use the connection handle if `mysql_store_result()` returned `NULL` (and thus you have no result set pointer). In this case, you can call `mysql_field_count()` to determine whether or not `mysql_store_result()` should have produced a non-empty result. This allows the client program to take proper action without knowing whether or not the query was a `SELECT` (or `SELECT`-like) statement. The example shown below illustrates how this may be done.

See Section 20.4.51 [`NULL mysql_store_result()`], page 365.

Return values

An unsigned integer representing the number of fields in a result set.

Errors

None.

Example

```
MYSQL_RES *result;
unsigned int num_fields;
unsigned int num_rows;
```

```
if (mysql_query(&mysql,query_string))
{
    // error
}
else // query succeeded, process any data returned by it
{
    result = mysql_store_result(&mysql);
    if (result)  // there are rows
    {
        num_fields = mysql_num_fields(result);
        // retrieve rows, then call mysql_free_result(result)
    }
    else  // mysql_store_result() returned nothing; should it have?
    {
        if (mysql_errno(&mysql))
{
           fprintf(stderr, "Error: %s\n", mysql_error(&mysql));
}
        else if (mysql_field_count(&mysql) == 0)
        {
            // query does not return data
            // (it was not a SELECT)
            num_rows = mysql_affected_rows(&mysql);
        }
    }
}
```

An alternative (if you KNOW that your query should have returned a result set) is to replace the `mysql_errno(&mysql)` call with a check if `mysql_field_count(&mysql)` is = 0. This will only happen if something went wrong.

20.4.36 `mysql_num_rows()`

`my_ulonglong mysql_num_rows(MYSQL_RES *result)`

Description

Returns the number of rows in the result set.

The use of `mysql_num_rows()` depends on whether you use `mysql_store_result()` or `mysql_use_result()` to return the result set. If you use `mysql_store_result()`, `mysql_num_rows()` may be called immediately. If you use `mysql_use_result()`, `mysql_num_rows()` will not return the correct value until all the rows in the result set have been retrieved.

Return values

The number of rows in the result set.

Errors

None.

20.4.37 mysql_options()

```
int mysql_options(MYSQL *mysql, enum mysql_option option, const char *arg)
```

Description

Can be used to set extra connect options and affect behavior for a connection.

Should be called after `mysql_init()` and before `mysql_connect()` or `mysql_real_connect()`.

The `option` argument is the option that you want to set; The `arg` argument is the value for the option. If the option is an integer, then `arg` should point to the value of the integer.

Possible options values:

Option	Argument type	Function
`MYSQL_OPT_CONNECT_TIMEOUT`	unsigned int *	Connect timeout in seconds.
`MYSQL_OPT_COMPRESS`	Not used	Use the compressed client/server protocol.
`MYSQL_OPT_NAMED_PIPE`	Not used	Use named pipes to connect to a **MySQL** server on NT.
`MYSQL_INIT_COMMAND`	char *	Command to execute when connecting to **MySQL** server. Will automatically be re-executed when reconnecting.
`MYSQL_READ_DEFAULT_FILE`	char *	Read options from the named option file instead of from '`my.cnf`'.
`MYSQL_READ_DEFAULT_GROUP`	char *	Read options from the named group from '`my.cnf`'.or the file specified with `MYSQL_READ_DEFAULT_FILE`.

Note that the group `client` is always read if you use `MYSQL_READ_DEFAULT_FILE` or `MYSQL_READ_DEFAULT_GROUP`.

The specified group in the option file may contain the following options:

compress	Use the compressed client/server protocol.
database	Connect to this database if there was no database in the connect command
debug	Debug options
host	Default host name
init-command	Command to execute when connecting to **MySQL** server. Will automatically be re-executed when reconnecting.
password	Default password
pipe	Use named pipes to connect to a **MySQL** server on NT.
port	Default port number
return-found-rows	Tell `mysql_info()` to return found rows instead of updated rows when using `UPDATE`.

socket	Default socket number
timeout	Connect timeout in seconds.
user	Default user

For more information about option files, see Section 4.15.4 [Option files], page 79.

Return values

Zero for success. Non-zero if you used an unknown option.

Example

```
MYSQL mysql;

mysql_init(&mysql);
mysql_options(&mysql,MYSQL_OPT_COMPRESS,0);
mysql_options(&mysql,MYSQL_READ_DEFAULT_GROUP,"odbc");
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

The above requests the client to use the compressed client/server protocol and read the additional options from the `odbc` section in the `my.cnf` file.

20.4.38 `mysql_ping()`

```
int mysql_ping(MYSQL *mysql)
```

Description

Checks whether or not the connection to the server is working. If it has gone down, an automatic reconnection is attempted.

This function can be used by clients that remain idle for a long while, to check whether or not the server has closed the connection and reconnect if necessary.

Return values

Zero if the server is alive. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.39 mysql_query()

```
int mysql_query(MYSQL *mysql, const char *query)
```

Description

Executes the SQL query pointed to by the null-terminated string `query`. The query must consist of a single SQL statement. You should not add a terminating semicolon (';') or `\g` to the statement.

`mysql_query()` cannot be used for queries that contain binary data; you should use `mysql_real_query()` instead. (Binary data may contain the '`\0`' character, which `mysql_query()` interprets as the end of the query string.)

Return values

Zero if the query was successful. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.40 mysql_real_connect()

```
MYSQL *mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
const char *passwd, const char *db, unsigned int port, const char *unix_socket,
unsigned int client_flag)
```

Description

`mysql_real_connect()` attempts to establish a connection to a **MySQL** database engine running on `host`. `mysql_real_connect()` must complete successfully before you can execute any of the other API functions, with the exception of `mysql_get_client_info()`.

The parameters are specified as follows:

- `mysql` is a pointer to a `MYSQL` connection structure, or `NULL`.

  If `mysql` is `NULL`, the C API allocates memory for the connection structure automatically and frees it when you call `mysql_close()`. The disadvantage of this approach

is that you can't retrieve an error message if the connection fails. (To get error information from `mysql_errno()` or `mysql_error()`, you must provide a valid `MYSQL` pointer.)

If the first parameter is not `NULL`, it should be the address of an existing `MYSQL` structure. In this case, before calling `mysql_real_connect()` you must call `mysql_init()` to initialize the `MYSQL` structure. See the example below.

- The value of `host` may be either a hostname or an IP address. If `host` is `NULL` or the string `"localhost"`, a connection to the local host is assumed. If the OS supports sockets (Unix) or named pipes (Win32), they are used instead of TCP/IP to connect to the server.
- The `user` parameter contains the user's **MySQL** login ID. If `user` is `NULL`, the current user is assumed. Under Unix, this is the current login name. Under Windows ODBC, the current user name must be specified explicitly. See Section 16.4 [ODBC administrator], page 304.
- The `passwd` parameter contains the password for `user`. If `passwd` is `NULL`, only entries in the `user` table for the user that have a blank password field will be checked for a match. This allows the database administrator to set up the **MySQL** privilege system in such a way that users get different privileges depending on whether or not they have specified a password.

  Note: Do not attempt to encrypt the password before calling `mysql_real_connect()`; password encryption is handled automatically by the client API.
- `db` is the database name. If `db` is not `NULL`, the connection will set the default database to this value.
- If `port` is not 0, the value will be used as the port number for the TCP/IP connection. Note that the `host` parameter determines the type of the connection.
- If `unix_socket` is not `NULL`, the string specifies the socket or named pipe that should be used. Note that the `host` parameter determines the type of the connection.
- The value of `client_flag` is usually 0, but can be set to a combination of the following flags in very special circumstances:

Flag name	**Flag meaning**
`CLIENT_FOUND_ROWS`	Return the number of found rows, not the number of affected rows
`CLIENT_NO_SCHEMA`	Don't allow the `db_name.tbl_name.col_name` syntax. This is for ODBC; it causes the parser to generate an error if you use that syntax, which is is useful for trapping bugs in some ODBC programs.
`CLIENT_COMPRESS`	Use compression protocol
`CLIENT_ODBC`	The client is an ODBC client. This changes `mysqld` to be more ODBC-friendly.

Return values

A `MYSQL*` connection handle if the connection was successful. `NULL` if the connection was unsuccessful. For a successful connection, the return value is the same as the value of the first parameter, unless you pass `NULL` for that parameter.

Errors

`CR_CONN_HOST_ERROR`
: Failed to connect to the **MySQL** server.

`CR_CONNECTION_ERROR`
: Failed to connect to the local **MySQL** server.

`CR_IPSOCK_ERROR`
: Failed to create an IP socket.

`CR_OUT_OF_MEMORY`
: Out of memory.

`CR_SOCKET_CREATE_ERROR`
: Failed to create a Unix socket.

`CR_UNKNOWN_HOST`
: Failed to find the IP address for the hostname.

`CR_VERSION_ERROR`
: A protocol mismatch resulted from attempting to connect to a server with a client library that uses a different protocol version. This can happen if you use a very old client library to connect to a new server that wasn't started with the `--old-protocol` option.

`CR_NAMEDPIPEOPEN_ERROR;`
: Failed to create a named pipe on Win32.

`CR_NAMEDPIPEWAIT_ERROR;`
: Failed to wait for a named pipe on Win32.

`CR_NAMEDPIPESETSTATE_ERROR;`
: Failed to get a pipe handler on Win32.

Example

```
MYSQL mysql;

mysql_init(&mysql);
if (!mysql_real_connect(&mysql,"host","user","passwd","database",0,NULL,0))
{
    fprintf(stderr, "Failed to connect to database: Error: %s\n",
          mysql_error(&mysql));
}
```

20.4.41 mysql_real_query()

```
int mysql_real_query(MYSQL *mysql, const char *query, unsigned int length)
```

Description

Executes the SQL query pointed to by query, which should be a string `length` bytes long. The query must consist of a single SQL statement. You should not add a terminating semicolon (';') or `\g` to the statement.

You *must* use `mysql_real_query()` rather than `mysql_query()` for queries that contain binary data, since binary data may contain the '`\0`' character. In addition, `mysql_real_query()` is faster than `mysql_query()` since it does not call `strlen()` on the query string.

Return values

Zero if the query was successful. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.42 mysql_reload()

```
int mysql_reload(MYSQL *mysql)
```

Description

Asks the **MySQL** server to reload the grant tables. The connected user must have the **reload** privilege.

This function is deprecated. It is preferable to use `mysql_query()` to issue a SQL `FLUSH PRIVILEGES` statement instead.

Return values

Zero for success. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.43 `mysql_row_seek()`

```
MYSQL_ROW_OFFSET mysql_row_seek(MYSQL_RES *result, MYSQL_ROW_OFFSET offset)
```

Description

Sets the row cursor to an arbitrary row in a query result set. This requires that the result set structure contains the entire result of the query, so `mysql_row_seek()` may be used in conjunction only with `mysql_store_result()`, not with `mysql_use_result()`.

The offset should be a value returned from a call to `mysql_row_tell()` or to `mysql_row_seek()`. This value is not simply a row number; if you want to seek to a row within a result set using a row number, use `mysql_data_seek()` instead.

Return values

The previous value of the row cursor. This value may be passed to a subsequent call to `mysql_row_seek()`.

Errors

None.

20.4.44 `mysql_row_tell()`

```
MYSQL_ROW_OFFSET mysql_row_tell(MYSQL_RES *result)
```

Description

Returns the current position of the row cursor for the last `mysql_fetch_row()`. This value can be used as an argument to `mysql_row_seek()`.

You should use `mysql_row_tell()` only after `mysql_store_result()`, not after `mysql_use_result()`.

Return values

The current offset of the row cursor.

Errors

None.

20.4.45 mysql_select_db()

```
int mysql_select_db(MYSQL *mysql, const char *db)
```

Description

Causes the database specified by `db` to become the default (current) database on the connection specified by `mysql`. In subsequent queries, this database is the default for table references that do not include an explicit database specifier.

`mysql_select_db()` fails unless the connected user can be authenticated as having permission to use the database.

Return values

Zero for success. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.46 mysql_shutdown()

```
int mysql_shutdown(MYSQL *mysql)
```

Description

Asks the database server to shutdown. The connected user must have **shutdown** privileges.

Return values

Zero for success. Non-zero if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.47 `mysql_stat()`

```
char *mysql_stat(MYSQL *mysql)
```

Description

Returns a character string containing information similar to that provided by the `mysqladmin status` command. This includes uptime in seconds and the number of running threads, questions, reloads and open tables.

Return values

A character string describing the server status. `NULL` if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.48 `mysql_store_result()`

```
MYSQL_RES *mysql_store_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query which successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`mysql_store_result()` reads the entire result of a query to the client, allocates a `MYSQL_RES` structure, and places the result into this structure.

An empty result set is returned if there are no rows returned. (An empty result set differs from a `NULL` return value.)

Once you have called `mysql_store_result()`, you may call `mysql_num_rows()` to find out how many rows are in the result set.

You can call `mysql_fetch_row()` to fetch rows from the result set, or `mysql_row_seek()` and `mysql_row_tell()` to obtain or set the current row position within the result set.

You must call `mysql_free_result()` once you are done with the result set.

See Section 20.4.51 [`NULL mysql_store_result()`], page 365.

Return values

A `MYSQL_RES` result structure with the results. `NULL` if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_OUT_OF_MEMORY`
: Out of memory.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.49 `mysql_thread_id()`

```
unsigned long mysql_thread_id(MYSQL *mysql)
```

Description

Returns the thread ID of the current connection. This value can be used as an argument to `mysql_kill()` to kill the thread.

If the connection is lost and you reconnect with `mysql_ping()`, the thread ID will change. This means you should not get the thread ID and store it for later, you should get it when you need it.

Return values

The thread ID of the current connection.

Errors

None.

20.4.50 mysql_use_result()

```
MYSQL_RES *mysql_use_result(MYSQL *mysql)
```

Description

You must call `mysql_store_result()` or `mysql_use_result()` for every query which successfully retrieves data (`SELECT`, `SHOW`, `DESCRIBE`, `EXPLAIN`).

`mysql_use_result()` initiates a result set retrieval but does not actually read the result set into the client like `mysql_store_result()` does. Instead, each row must be retrieved individually by making calls to `mysql_fetch_row()`. This reads the result of a query directly from the server without storing it in a temporary table or local buffer, which is somewhat faster and uses much less memory than `mysql_store_result()`. The client will only allocate memory for the current row and a communication buffer that may grow up to `max_allowed_packet` bytes.

On the other hand, you shouldn't use `mysql_use_result()` if you are doing a lot of processing for each row on the client side, or if the output is sent to a screen on which the user may type a `^S` (stop scroll). This will tie up the server and prevent other threads from updating any tables from which the data are fetched.

When using `mysql_use_result()`, you must execute `mysql_fetch_row()` until a `NULL` value is returned, otherwise the unfetched rows will be returned as part of the result set for your next query. The C API will give the error `Commands out of sync; You can't run this command now` if you forget to do this!

You may not use `mysql_data_seek()`, `mysql_row_seek()`, `mysql_row_tell()`, `mysql_num_rows()` or `mysql_affected_rows()` with a result returned from `mysql_use_result()`, nor may you issue other queries until the `mysql_use_result()` has finished. (However, after you have fetched all the rows, `mysql_num_rows()` will accurately return the number of rows fetched.)

You must call `mysql_free_result()` once you are done with the result set.

Return values

A `MYSQL_RES` result structure. `NULL` if an error occurred.

Errors

`CR_COMMANDS_OUT_OF_SYNC`
: Commands were executed in an improper order.

`CR_OUT_OF_MEMORY`
: Out of memory.

`CR_SERVER_GONE_ERROR`
: The **MySQL** server has gone away.

`CR_SERVER_LOST`
: The connection to the server was lost during the query.

`CR_UNKNOWN_ERROR`
: An unknown error occurred.

20.4.51 Why is it that after `mysql_query()` returns success, `mysql_store_result()` sometimes returns `NULL`?

It is possible for `mysql_store_result()` to return `NULL` following a successful call to `mysql_query()`. When this happens, it means one of the following conditions occurred:

- There was a `malloc()` failure (for example, if the result set was too large).
- The data couldn't be read (an error occurred on the connection).
- The query returned no data (e.g., it was an `INSERT`, `UPDATE` or `DELETE`).

You can always check whether or not the statement should have produced a non-empty result by calling `mysql_field_count()`. If `mysql_field_count()` returns zero, the result is empty and the last query was a statement that does not return values (for example, an `INSERT` or a `DELETE`). If `mysql_field_count()` returns a non-zero value, the statement should have produced a non-empty result. See the description of the `mysql_field_count()` function for an example.

You can test for an error by calling `mysql_error()` or `mysql_errno()`.

20.4.52 What results can I get from a query?

In addition to the result set returned by a query, you can also get the following information:

- `mysql_affected_rows()` returns the number of rows affected by the last query when doing an `INSERT`, `UPDATE` or `DELETE`. An exception is that if `DELETE` is used without a `WHERE` clause, the table is truncated, which is much faster! In this case, `mysql_affected_rows()` returns zero for the number of records affected.
- `mysql_num_rows()` returns the number of rows in a result set. With `mysql_store_result()`, `mysql_num_rows()` may be called as soon as `mysql_store_result()` returns. With `mysql_use_result()`, `mysql_num_rows()` may be called only after you have fetched all the rows with `mysql_fetch_row()`.
- `mysql_insert_id()` returns the ID generated by the last query that inserted a row into a table with an `AUTO_INCREMENT` index. See Section 20.4.29 [`mysql_insert_id()`], page 348.
- Some queries (`LOAD DATA INFILE ...`, `INSERT INTO ... SELECT ...`, `UPDATE`) return additional info. The result is returned by `mysql_info()`. See the description for `mysql_info()` for the format of the string that it returns. `mysql_info()` returns a `NULL` pointer if there is no additional information.

20.4.53 How can I get the unique ID for the last inserted row?

If you insert a record in a table containing a column that has the `AUTO_INCREMENT` attribute, you can get the most recently generated ID by calling the `mysql_insert_id()` function.

You can also retrieve the ID by using the `LAST_INSERT_ID()` function in a query string that you pass to `mysql_query()`.

You can check if an `AUTO_INCREMENT` index is used by executing the following code. This also checks if the query was an `INSERT` with an `AUTO_INCREMENT` index:

```
if (mysql_error(&mysql)[0] == 0 &&
    mysql_num_fields(result) == 0 &&
    mysql_insert_id(&mysql) != 0)
{
    used_id = mysql_insert_id(&mysql);
}
```

The most recently generated ID is maintained in the server on a per-connection basis. It will not be changed by another client. It will not even be changed if you update another `AUTO_INCREMENT` column with a non-magic value (that is, a value that is not `NULL` and not `0`).

If you want to use the ID that was generated for one table and insert it into a second table, you can use SQL statements like this:

```
INSERT INTO foo (auto,text)
    VALUES(NULL,'text');              # generate ID by inserting NULL
INSERT INTO foo2 (id,text)
    VALUES(LAST_INSERT_ID(),'text');  # use ID in second table
```

20.4.54 Problems linking with the C API

When linking with the C API, the following errors may occur on some systems:

```
gcc -g -o client test.o -L/usr/local/lib/mysql -lmysqlclient -lsocket -lnsl

Undefined        first referenced
 symbol          in file
floor            /usr/local/lib/mysql/libmysqlclient.a(password.o)
ld: fatal: Symbol referencing errors. No output written to client
```

If this happens on your system, you must include the math library by adding `-lm` to the end of the compile/link line.

20.4.55 How to make a thread-safe client

The client is "almost" thread-safe. The biggest problem is that the subroutines in '`net.c`' that read from sockets are not interrupt-safe. This was done with the thought that you might want to have your own alarm that can break a long read to a server.

The standard client libraries are not compiled with the thread options.

To get a thread-safe client, use the `-lmysys`, `-lstring` and `-ldbug` libraries and `net_serv.o` that the server uses.

When using a threaded client, you can make great use of the routines in the 'thr_alarm.c' file. If you are using routines from the mysys library, the only thing you must remember is to call my_init() first!

All functions except mysql_real_connect() are currently thread-safe. The following notes describe how to compile a thread-safe client library and use it in a thread-safe manner. (The notes below for mysql_real_connect() actually apply to mysql_connect() as well, but since mysql_connect() is deprecated, you should be using mysql_real_connect() anyway.)

To make mysql_real_connect() thread-safe, you must recompile the client library with this command:

```
shell> CPPFLAGS=-DTHREAD_SAFE_CLIENT ./configure ...
```

You may get some errors because of undefined symbols when linking the standard client, because the pthread libraries are not included by default.

The resulting 'libmysqlclient.a' library is now thread-safe. What this means is that client code is thread-safe as long as two threads don't query the same connection handle returned by mysql_real_connect() at the same time; the client/server protocol allows only one request at a time on a given connection. If you want to use multiple threads on the same connection, you must have a mutex lock around your mysql_query() and mysql_store_result() call combination. Once mysql_store_result() is ready, the lock can be released and other threads may query the same connection. (In other words, different threads can use different MYSQL_RES pointers that were created with mysql_store_result(), as long as they use the proper locking protocol.) If you program with POSIX threads, you can use pthread_mutex_lock() and pthread_mutex_unlock() to establish and release a mutex lock.

If you used mysql_use_result() rather than mysql_store_result(), the lock would need to surround mysql_use_result() and the calls to mysql_fetch_row(). However, it really is best for threaded clients not to use mysql_use_result().

20.5 MySQL Perl API

This section documents the Perl DBI interface. The former interface was called mysqlperl. Since DBI/DBD now is the recommended Perl interface, mysqlperl is obsolete and is not documented here.

20.5.1 DBI with DBD::mysql

DBI is a generic interface for many databases. That means that you can write a script that works with many different database engines without change. You need a DataBase Driver (DBD) defined for each database type. For **MySQL**, this driver is called DBD::mysql.

For more information on the Perl5 DBI, please visit the DBI web page and read the documentation:

http://www.symbolstone.org/technology/perl/DBI/index.html

For more information on Object Oriented Programming (OOP) as defined in Perl5, see the Perl OOP page:

http://language.perl.com/info/documentation.html

Installation instructions for **MySQL** Perl support are given in Section 4.10 [Perl support], page 48.

20.5.2 The DBI interface

Portable DBI methods

`connect`	Establishes a connection to a database server
`disconnect`	Disconnects from the database server
`prepare`	Prepares a SQL statement for execution
`execute`	Executes prepared statements
`do`	Prepares and executes a SQL statement
`quote`	Quotes string or `BLOB` values to be inserted
`fetchrow_array`	Fetches the next row as an array of fields.
`fetchrow_arrayref`	Fetches next row as a reference array of fields
`fetchrow_hashref`	Fetches next row as a reference to a hashtable
`fetchall_arrayref`	Fetches all data as an array of arrays
`finish`	Finishes a statement and let the system free resources
`rows`	Returns the number of rows affected
`data_sources`	Returns an array of databases available on localhost
`ChopBlanks`	Controls whether `fetchrow_*` methods trim spaces
`NUM_OF_PARAMS`	The number of placeholders in the prepared statement
`NULLABLE`	Which columns can be `NULL`
`trace`	Perform tracing for debugging

MySQL-specific methods

`insertid`	The latest `AUTO_INCREMENT` value
`is_blob`	Which column are `BLOB` values
`is_key`	Which columns are keys
`is_num`	Which columns are numeric
`is_pri_key`	Which columns are primary keys
`is_not_null`	Which columns CANNOT be `NULL`. See `NULLABLE`.
`length`	Maximum possible column sizes
`max_length`	Maximum column sizes actually present in result
`NAME`	Column names
`NUM_OF_FIELDS`	Number of fields returned
`table`	Table names in returned set
`type`	All column types

The Perl methods are described in more detail in the following sections. Variables used for method return values have these meanings:

`$dbh`	Database handle
`$sth`	Statement handle
`$rc`	Return code (often a status)
`$rv`	Return value (often a row count)

Portable DBI methods

`connect($data_source, $username, $password)`

Use the `connect` method to make a database connection to the data source. The `$data_source` value should begin with `DBI:driver_name:`. Example uses of `connect` with the `DBD::mysql` driver:

```
$dbh = DBI->connect("DBI:mysql:$database", $user, $password);
$dbh = DBI->connect("DBI:mysql:$database:$hostname",
                    $user, $password);
$dbh = DBI->connect("DBI:mysql:$database:$hostname:$port",
                    $user, $password);
```

If the user name and/or password are undefined, `DBI` uses the values of the `DBI_USER` and `DBI_PASS` environment variables, respectively. If you don't specify a hostname, it defaults to `'localhost'`. If you don't specify a port number, it defaults to the default **MySQL** port (3306).

As of `Msql-Mysql-modules` version 1.2009, the `$data_source` value allows certain modifiers:

`mysql_read_default_file=file_name`

Read '`filename`' as an option file. For information on option files, see Section 4.15.4 [Option files], page 79.

`mysql_read_default_group=group_name`

The default group when reading an option file is normally the `[client]` group. By specifying the `mysql_read_default_group` option, the default group becomes the `[group_name]` group.

`mysql_compression=1`

Use compressed communication between the client and server (**MySQL** 3.22.3 or later).

`mysql_socket=/path/to/socket`

Specify the pathname of the Unix socket that is used to connect to the server (**MySQL** 3.21.15 or later).

Multiple modifiers may be given; each must be preceded by a semicolon.

For example, if you want to avoid hardcoding the user name and password into a `DBI` script, you can take them from the user's '`~/.my.cnf`' option file instead by writing your `connect` call like this:

```
$dbh = DBI->connect("DBI:mysql:$database"
                    . ";mysql_read_default_file=$ENV{HOME}/.my.cnf",
                    $user, $password);
```

This call will read options defined for the `[client]` group in the option file. If you wanted to do the same thing, but use options specified for the `[perl]` group instead, you could use this:

```
$dbh = DBI->connect("DBI:mysql:$database"
                    . ";mysql_read_default_file=$ENV{HOME}/.my.cnf"
                    . ";mysql_read_default_group=perl",
                    $user, $password);
```

`disconnect`

The `disconnect` method disconnects the database handle from the database. This is typically called right before you exit from the program. Example:

```
$rc = $dbh->disconnect;
```

`prepare($statement)`

Prepares a SQL statement for execution by the database engine and returns a statement handle (`$sth`) which you can use to invoke the `execute` method. Typically you handle `SELECT` statements (and `SELECT`-like statements such as `SHOW`, `DESCRIBE` and `EXPLAIN`) by means of `prepare` and `execute`. Example:

```
$sth = $dbh->prepare($statement)
    or die "Can't prepare $statement: $dbh->errstr\n";
```

`execute` The `execute` method executes a prepared statement. For non-`SELECT` statements, `execute` returns the number of rows affected. If no rows are affected, `execute` returns `"0E0"`, which Perl treats as zero but regards as true. For `SELECT` statements, `execute` only starts the SQL query in the database; you need to use one of the `fetch_*` methods described below to retrieve the data. Example:

```
$rv = $sth->execute
          or die "can't execute the query: $sth->errstr;
```

`do($statement)`

The `do` method prepares and executes a SQL statement and returns the number of rows affected. If no rows are affected, `do` returns `"0E0"`, which Perl treats as zero but regards as true. This method is generally used for non-`SELECT` statements which cannot be prepared in advance (due to driver limitations) or which do not need to executed more than once (inserts, deletes, etc.). Example:

```
$rv = $dbh->do($statement)
        or die "Can't execute $statement: $dbh- >errstr\n";
```

`quote($string)`

The `quote` method is used to "escape" any special characters contained in the string and to add the required outer quotation marks. Example:

```
$sql = $dbh->quote($string)
```

`fetchrow_array`

This method fetches the next row of data and returns it as an array of field values. Example:

```
while(@row = $sth->fetchrow_array) {
        print qw($row[0]\t$row[1]\t$row[2]\n);
}
```

`fetchrow_arrayref`

This method fetches the next row of data and returns it as a reference to an array of field values. Example:

```
while($row_ref = $sth->fetchrow_arrayref) {
        print qw($row_ref->[0]\t$row_ref->[1]\t$row_ref->[2]\n);
}
```

`fetchrow_hashref`
: This method fetches a row of data and returns a reference to a hash table containing field name/value pairs. This method is not nearly as efficient as using array references as demonstrated above. Example:

```
while($hash_ref = $sth->fetchrow_hashref) {
        print qw($hash_ref->{firstname}\t$hash_ref->{lastname}\t\
                $hash_ref- > title}\n);
}
```

`fetchall_arrayref`
: This method is used to get all the data (rows) to be returned from the SQL statement. It returns a reference to an array of arrays of references to each row. You access or print the data by using a nested loop. Example:

```
my $table = $sth->fetchall_arrayref
                or die "$sth->errstr\n";
my($i, $j);
for $i ( 0 .. $#{$table} ) {
        for $j ( 0 .. $#{$table->[$i]} ) {
                print "$table->[$i][$j]\t";
        }
        print "\n";
}
```

`finish`
: Indicates that no more data will be fetched from this statement handle. You call this method to free up the statement handle and any system resources it may be holding. Example:

```
$rc = $sth->finish;
```

`rows`
: Returns the number of rows changed (updated, deleted, etc.) by the last command. This is usually used after a non-`SELECT` `execute` statement. Example:

```
$rv = $sth->rows;
```

`NULLABLE`
: Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that this column may contain `NULL` values. Example:

```
$null_possible = $sth->{NULLABLE};
```

`NUM_OF_FIELDS`
: This attribute indicates the number of fields returned by a `SELECT` or `SHOW FIELDS` statement. You may use this for checking whether a statement returned a result: A zero value indicates a non-`SELECT` statement like `INSERT`, `DELETE` or `UPDATE`. Example:

```
$nr_of_fields = $sth->{NUM_OF_FIELDS};
```

`data_sources($driver_name)`
: This method returns an array containing names of databases available to the **MySQL** server on the host `'localhost'`. Example:

```
@dbs = DBI->data_sources("mysql");
```

`ChopBlanks`
: This attribute determines whether the `fetchrow_*` methods will chop leading and trailing blanks from the returned values. Example:

```
$sth->{'ChopBlanks'} =1;
```

`trace($trace_level)`
`trace($trace_level, $trace_filename)`
: The `trace` method enables or disables tracing. When invoked as a `DBI` class method, it affects tracing for all handles. When invoked as a database or statement handle method, it affects tracing for the given handle (and any future children of the handle). Setting `$trace_level` to 2 provides detailed trace information. Setting `$trace_level` to 0 disables tracing. Trace output goes to the standard error output by default. If `$trace_filename` is specified, the file is opened in append mode and output for *all* traced handles is written to that file. Example:

```
DBI->trace(2);                 # trace everything
DBI->trace(2,"/tmp/dbi.out"); # trace everything to /tmp/dbi.out
$dth->trace(2);                # trace this database handle
$sth->trace(2);                # trace this statement handle
```

You can also enable `DBI` tracing by setting the `DBI_TRACE` environment variable. Setting it to a numeric value is equivalent to calling `DBI->(value)`. Setting it to a pathname is equivalent to calling `DBI->(2,value)`.

MySQL-specific methods

The methods shown below are **MySQL**-specific and not part of the `DBI` standard. Several of them are now deprecated: `is_blob`, `is_key`, `is_num`, `is_pri_key`, `is_not_null`, `length`, `max_length`, and `table`. Where `DBI`-standard alternatives exist, they are noted below.

`insertid` If you use the `AUTO_INCREMENT` feature of **MySQL**, the new auto-incremented values will be stored here. Example:

```
$new_id = $sth->{insertid};
```

As an alternative, you can use `$dbh->{'mysql_insertid'}`.

`is_blob` Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that the respective column is a `BLOB`. Example:

```
$keys = $sth->{is_blob};
```

`is_key` Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that the respective column is a key. Example:

```
$keys = $sth->{is_key};
```

`is_num` Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that the respective column contains numeric values. Example:

```
$nums = $sth->{is_num};
```

`is_pri_key`
: Returns a reference to an array of boolean values; for each element of the array, a value of TRUE indicates that the respective column is a primary key. Example:

```
$pri_keys = $sth->{is_pri_key};
```

`is_not_null`
: Returns a reference to an array of boolean values; for each element of the array, a value of FALSE indicates that this column may contain `NULL` values. Example:

```
$not_nulls = $sth->{is_not_null};
```

`is_not_null` is deprecated; it is preferable to use the `NULLABLE` attribute (described above), since that is a DBI standard.

`length`
`max_length`
: Each of these methods returns a reference to an array of column sizes. The `length` array indicates the maximum possible sizes that each column may be (as declared in the table description). The `max_length` array indicates the maximum sizes actually present in the result table. Example:

```
$lengths = $sth->{length};
$max_lengths = $sth->{max_length};
```

`NAME`
: Returns a reference to an array of column names. Example:

```
$names = $sth->{NAME};
```

`table`
: Returns a reference to an array of table names. Example:

```
$tables = $sth->{table};
```

`type`
: Returns a reference to an array of column types. Example:

```
$types = $sth->{type};
```

20.5.3 More `DBI/DBD` information

You can use the `perldoc` command to get more information about `DBI`.

```
perldoc DBI
perldoc DBI::FAQ
perldoc DBD::mysql
```

You can also use the `pod2man`, `pod2html`, etc., tools to translate to other formats.

And of course you can find the latest `DBI` information at the `DBI` web page:

http://www.symbolstone.org/technology/perl/DBI/index.html

20.6 MySQL Eiffel wrapper

The **MySQL** Contrib directory contains an Eiffel wrapper written by Michael Ravits.

You can also find this at: `http://www.netpedia.net/hosting/newplayer/`

20.7 MySQL Java connectivity (JDBC)

There are 2 supported JDBC drivers for MySQL (the twz and mm driver). You can find a copy of these at http://www.mysql.com/Contrib. For documentation consult any JDBC documentation and the drivers own documentation for **MySQL** specific features.

20.8 MySQL PHP API

PHP is a server-side, HTML embedded scripting language that may be used to create dynamic web pages. It contains support for accessing several databases, including **MySQL**. PHP may be run as a separate program, or compiled as a module for use with the Apache web server.

The distribution and documentation are available at the PHP website.

20.9 MySQL C++ APIs

Two API's are available in the **MySQL** Contrib directory.

20.10 MySQL Python APIs

The **MySQL** Contrib directory contains a Python interface written by Joseph Skinner.

You can also use the Python interface to iODBC to access a **MySQL** server. mxODBC

20.11 MySQL TCL APIs

TCL at binevolve. The Contrib directory contains a TCL interface that is based on msqltcl 1.50.

21 How MySQL compares to other databases

21.1 How MySQL compares to mSQL

This section has been written by the **MySQL** developers, so it should be read with that in mind. But there are NO factual errors that we know of.

For a list of all supported limits, functions and types, see the `crash-me` web page.

Performance

For a true comparison of speed, consult the growing **MySQL** benchmark suite. See Chapter 11 [Benchmarks], page 265.

Because there is no thread creation overhead, a small parser, few features and simple security, `mSQL` should be quicker at:

- Tests that perform repeated connects and disconnects, running a very simple query during each connection.
- `INSERT` operations into very simple tables with few columns and keys.
- `CREATE TABLE` and `DROP TABLE`.
- `SELECT` on something that isn't an index. (A table scan is very easy.)

Since these operations are so simple, it is hard to be better at them when you have a higher startup overhead. After the connection is established, **MySQL** should perform much better.

On the other hand, **MySQL** is much faster than `mSQL` (and most other SQL implementions) on the following:

- Complex `SELECT` operations.
- Retrieving large results (**MySQL** has a better, faster and safer protocol).
- Tables with variable-length strings, since **MySQL** has more efficent handling and can have indexes on `VARCHAR` columns.
- Handling tables with many columns.
- Handling tables with large record lengths.
- `SELECT` with many expressions.
- `SELECT` on large tables.
- Handling many connections at the same time. **MySQL** is fully multi-threaded. Each connection has its own thread, which means that no thread has to wait for another (unless a thread is modifying a table another thread wants to access.) In `mSQL`, once one connection is established, all others must wait until the first has finished, regardless of whether the connection is running a query that is short or long. When the first connection terminates, the next can be served, while all the others wait again, etc.
- Joins. `mSQL` can become pathologically slow if you change the order of tables in a `SELECT`. In the benchmark suite, a time more than 15000 times slower than **MySQL** was seen. This is due to `mSQL`'s lack of a join optimizer to

order tables in the optimal order. However, if you put the tables in exactly the right order in mSQL2 and the WHERE is simple and uses index columns, the join will be relatively fast! See Chapter 11 [Benchmarks], page 265.

- ORDER BY and GROUP BY.
- DISTINCT.
- Using TEXT or BLOB columns.

SQL Features

- GROUP BY and HAVING. mSQL does not support GROUP BY at all. **MySQL** supports a full GROUP BY with both HAVING and the following functions: COUNT(), AVG(), MIN(), MAX(), SUM() and STD(). COUNT(*) is optimized to return very quickly if the SELECT retrieves from one table, no other columns are retrieved and there is no WHERE clause. MIN() and MAX() may take string arguments.
- INSERT and UPDATE with calculations. **MySQL** can do calculations in an INSERT or UPDATE. For example:

```
mysql> UPDATE SET x=x*10+y WHERE x<20;
```

- Aliasing. **MySQL** has column aliasing.
- Qualifying column names. In **MySQL**, if a column name is unique among the tables used in a query, you do not have to use the full qualifier.
- SELECT with functions. **MySQL** has many functions (too many to list here; see Section 7.3 [Functions], page 136).

Disk space efficiency

That is, how small can you make your tables?

MySQL has very precise types, so you can create tables that take very little space. An example of a useful **MySQL** datatype is the MEDIUMINT that is 3 bytes long. If you have 100,000,000 records, saving even one byte per record is very important.

mSQL2 has a more limited set of column types, so it is more difficult to get small tables.

Stability

This is harder to judge objectively. For a discussion of **MySQL** stability, see Section 1.5 [Stability], page 5.

We have no experience with mSQL stability, so we cannot say anything about that.

Price

Another important issue is the license. **MySQL** has a more flexible license than mSQL, and is also less expensive than mSQL. Whichever product you choose to use, remember to at least consider paying for a license or email support. (You are required to get a license if you include **MySQL** with a product that you sell, of course.)

Perl interfaces

MySQL has basically the same interfaces to Perl as mSQL with some added features.

JDBC (Java)

MySQL currently has 4 JDBC drivers:

- The gwe driver: A Java interface by GWE technologies (not supported anymore).
- The jms driver: An improved gwe driver by Xiaokun Kelvin ZHU X.Zhu@brad.ac.uk.
- The twz driver: A type 4 JDBC driver by Terrence W. Zellers zellert@voicenet.com. This is commercial but is free for private and educational use.
- The mm driver: A type 4 JDBC driver by Mark Matthews mmatthew@ecn.purdue.edu. This is released under the GPL.

The recommended drivers are the twz or mm driver. Both are reported to work excellently.

We know that `mSQL` has a JDBC driver, but we have too little experience with it to compare.

Rate of development

MySQL has a very small team of developers, but we are quite used to coding C and C++ very rapidly. Since threads, functions, `GROUP BY` and so on are still not implemented in `mSQL`, it has a lot of catching up to do. To get some perspective on this, you can view the `mSQL` '`HISTORY`' file for the last year and compare it with the News section of the **MySQL** Reference Manual (see Appendix D [News], page 400). It should be pretty obvious which one has developed most rapidly.

Utility programs

Both `mSQL` and **MySQL** have many interesting third-party tools. Since it is very easy to port upward (from `mSQL` to **MySQL**), almost all the interesting applications that are available for `mSQL` are also available for **MySQL**.

MySQL comes with a simple `msql2mysql` program that fixes differences in spelling between `mSQL` and **MySQL** for the most-used C API functions. For example, it changes instances of `msqlConnect()` to `mysql_connect()`. Converting a client program from `mSQL` to **MySQL** usually takes a couple of minutes.

21.1.1 How to convert `mSQL` tools for MySQL

According to our experience, it would just take a few hours to convert tools such as `msqltcl` and `msqljava` that use the `mSQL` C API so that they work with the **MySQL** C API.

The conversion procedure is:

1. Run the shell script `msql2mysql` on the source. This requires the `replace` program, which is distributed with **MySQL**.
2. Compile.
3. Fix all compiler errors.

Differences between the mSQL C API and the **MySQL** C API are:

- **MySQL** uses a MYSQL structure as a connection type (mSQL uses an int).
- mysql_connect() takes a pointer to a MYSQL structure as a parameter. It is easy to define one globally or to use malloc() to get one. mysql_connect() also takes 2 parameters for specifying the user and password. You may set these to NULL, NULL for default use.
- mysql_error() takes the MYSQL structure as a parameter. Just add the parameter to your old msql_error() code if you are porting old code.
- **MySQL** returns an error number and a text error message for all errors. mSQL returns only a text error message.
- Some incompatibilities exist as a result of **MySQL** supporting multiple connections to the server from the same process.

21.1.2 How mSQL and MySQL client/server communications protocols differ

There are enough differences that it is impossible (or at least not easy) to support both.

The most significant ways in which the **MySQL** protocol differs from the mSQL protocol are listed below:

- A message buffer may contain many result rows.
- The message buffers are dynamically enlarged if the query or the result is bigger than the current buffer, up to a configurable server and client limit.
- All packets are numbered to catch duplicated or missing packets.
- All column values are sent in ASCII. The lengths of columns and rows are sent in packed binary coding (1, 2 or 3 bytes).
- **MySQL** can read in the result unbuffered (without having to store the full set in the client).
- If a single write/read takes more than 30 seconds, the server closes the connection.
- If a connection is idle for 8 hours, the server closes the connection.

21.1.3 How mSQL 2.0 SQL syntax differs from MySQL

Column types

MySQL Has the following additional types (among others; see see Section 7.6 [CREATE TABLE], page 167):

- ENUM type for one of a set of strings.
- SET type for many of a set of strings.
- BIGINT type for 64-bit integers.

MySQL also supports the following additional type attributes:

- UNSIGNED option for integer columns.
- ZEROFILL option for integer columns.

- `AUTO_INCREMENT` option for integer columns that are a `PRIMARY KEY`. See Section 20.4.29 [`mysql_insert_id()`], page 348.
- `DEFAULT` value for all columns.

`mSQL2` `mSQL` column types correspond to the **MySQL** types shown below:

`mSQL` type	Corresponding MySQL type
`CHAR(len)`	`CHAR(len)`
`TEXT(len)`	`TEXT(len)`. `len` is the maximal length. And `LIKE` works.
`INT`	`INT`. With many more options!
`REAL`	`REAL`. Or `FLOAT`. Both 4- and 8-byte versions are available.
`UINT`	`INT UNSIGNED`
`DATE`	`DATE`. Uses ANSI SQL format rather than `mSQL`'s own.
`TIME`	`TIME`
`MONEY`	`DECIMAL(12,2)`. A fixed-point value with two decimals.

Index creation

MySQL Indexes may be specified at table creation time with the `CREATE TABLE` statement.

`mSQL` Indexes must be created after the table has been created, with separate `CREATE INDEX` statements.

To insert a unique identifier into a table

MySQL Use `AUTO_INCREMENT` as a column type specifier. See Section 20.4.29 [`mysql_insert_id()`], page 348.

`mSQL` Create a `SEQUENCE` on a table and select the `_seq` column.

To obtain a unique identifier for a row

MySQL Add a `PRIMARY KEY` or `UNIQUE` key to the table.

`mSQL` Use the `_rowid` column. Observe that `_rowid` may change over time depending on many factors.

To get the time a column was last modified

MySQL Add a `TIMESTAMP` column to the table. This column is automatically set to the current date and time for `INSERT` or `UPDATE` statements if you don't give the column a value or if you give it a `NULL` value.

`mSQL` Use the `_timestamp` column.

`NULL` **value comparisons**

MySQL **MySQL** follows ANSI SQL and a comparison with `NULL` is always `NULL`.

`mSQL` In `mSQL`, `NULL = NULL` is TRUE. You must change `=NULL` to `IS NULL` and `<>NULL` to `IS NOT NULL` when porting old code from `mSQL` to **MySQL**.

String comparisons

MySQL Normally, string comparisons are performed in case-independent fashion with the sort order determined by the current character set (ISO-8859-1 Latin1 by default). If you don't like this, declare your columns with the `BINARY` attribute, which causes comparisons to be done according to the ASCII order used on the **MySQL** server host.

`mSQL` All string comparisons are performed in case-sensitive fashion with sorting in ASCII order.

Case-insensitive searching

MySQL `LIKE` is a case-insensitive or case-sensitive operator, depending on the columns involved. If possible, **MySQL** uses indexes if the `LIKE` argument doesn't start with a wildcard character.

`mSQL` Use `CLIKE`.

Handling of trailing spaces

MySQL Strips all spaces at the end of `CHAR` and `VARCHAR` columns. Use a `TEXT` column if this behavior is not desired.

`mSQL` Retains trailing space.

`WHERE` **clauses**

MySQL **MySQL** correctly prioritizes everything (`AND` is evaluated before `OR`). To get `mSQL` behavior in **MySQL**, use parentheses (as shown below).

`mSQL` Evaluates everything from left to right. This means that some logical calculations with more than three arguments cannot be expressed in any way. It also means you must change some queries when you upgrade to **MySQL**. You do this easily by adding parentheses. Suppose you have the following `mSQL` query:

```
mysql> SELECT * FROM table WHERE a=1 AND b=2 OR a=3 AND b=4;
```

To make **MySQL** evaluate this the way that `mSQL` would, you must add parentheses:

```
mysql> SELECT * FROM table WHERE (a=1 AND (b=2 OR (a=3 AND (b=4))));
```

Access control

MySQL Has tables to store grant (permission) options per user, host and database. See Section 6.6 [Privileges], page 98.

`mSQL` Has a file '`mSQL.acl`' in which you can grant read/write privileges for users.

21.2 How MySQL compares to PostgreSQL

`PostgreSQL` has some more advanced features like user-defined types, triggers, rules and some transaction support. However, PostgreSQL lacks many of the standard types and functions from ANSI SQL and ODBC. See the `crash-me` web page for a complete list of limits and which types and functions are supported or unsupported.

Normally, `PostgreSQL` is a magnitude slower than **MySQL**. See Chapter 11 [Benchmarks], page 265. This is due largely to their transactions system. If you really need transactions or the rich type system PostgreSQL offers and you can afford the speed penalty, you should take a look at PostgreSQL.

Appendix A Some MySQL users

A.1 General news sites

- A pro-Linux/tech news and comment/discussion site
- All about Linux
- 32Bits Online: because there's more than one way to compute
- New about new versions of computer related stuff
-

A.2 Some Web search engines

- AAA Matilda Web Search
- What's New
- Aladin
- Columbus Finder
- Spider
- Blitzsuche
- Indoseek Indonesia
- Yaboo - Yet Another BOOkmarker
- Yahoosuck
- OzSearch Internet Guide

A.3 Some Information search engines concentrated on some area

- Jobvertise: Post and search for jobs
- The Music Database
- Fotball (Soccer) search page
- TAKEDOWN - wrestling
- The International Lyrics Network
- Musicians looking for other musicians (Free Service)
- AddALL books searching and price comparison
- Harvard's Gray Herbarium Index of Plant Names
- The Game Development Search Engine
- My-Recipe.com; Cookbook at i-run.com
- The Innkeeper Vacation Guides
- The Mac Game Database uses PHP and MySQL
- Research Publications at Monash University in Australia
- OccupationalHealth & Safety website databse (a project for the ECC)
- Bioinformatics databases at the Montreal Children's Hospital using MySQL

A.4 Web sites the use MySQL as a backed

- Qt Widget and Object Repository
- Brazilian samba site (in Portuguese)
- Polish General Social Survey
- Expo2000 World-wide distribution of tickets for this event is implemented using MySQL and tcl/tk. More than 5000 travel-agencies all over the world have access to it.
- FreeVote.com is a free voting service with millions of users.

A.5 Some Domain/Internet/Web and related services

- Registry of Web providers that support **MySQL**
- Dynamic DNS Services
- Dynamic domain name service
- Open DNS Project; free dynamic DNS service
- Free 3rd level domains
- Online Database
- BigBiz Internet Services
- The Virt Gazette
- Global InfoNet Inc
- WebHosters - A Guide to WWW Providers
- Internet information server
- A technology news site
- WorldNet Communications - An Internet Services Provider
- Netizen: Australian-based web consultancy
- Search site for training courses in the UK
- Gannon Chat (GPL). Written in Perl and Javascript
- A general links directory
- A web-based bookmark management service
- Walnut Creek CDROM
- WWWThreads; Interactive discussion Forums
- In Italian; Storage data from meteo station
- Online "Person To Person" Auction
- Tips on web development
- Mailfriends.com is a FREE service for everybody who wants to find friends over the internet.
- Web Page Telnet BBS List
- UniNova Digital Postcards
- DSL providers search with reviews Made with MySQL and Modperl, all pages are generated dynamically out of the MySQL database

A.6 Web sites that use PHP and MySQL

- Jgaa's Internet - Official Support Site
- Ionline - online publication: **MySQL**, PHP, Java, Web programming, DB development
- BaBoo(Browse and bookmark). Free web-based bookmark manager and Calendar
- Course Schedule System at Pensacola Junior College
- Florida Community College at Jacksonville
- A beginners tutoral of how to start using **MySQL**
- 32bit.com; An extensive shareware / freeware archive
- Jokes 2000
- Burken.NU Burken is a webhotel that provides scripts, among other things, for remote users, like counters, guestbooks etc.
- tips.pair.com Contains tips on html, javascript, 2d/3d graphics and PHP3/MySQL. All pages are generated from a database.

A.7 Some MySQL consultants

- Ayni AG
- Online Database
- DataGuard (Uses **MySQL** and PHP)
- WWITS (Uses **MySQL** and PHP)
- WCN - The World Community Network
- Chip Castle Dot Com Inc
- Cybersource Pty. Ltd
- Spring infotainment gmbh & co. kg
- Develops websites using MySQL

A.8 Programming

- The Perl CPAN Testers results page

A.9 Uncategorized pages

- AZC.COM's Feature Showcase
- Course Search
- Northerbys Online Auctions
- Amsterdam Airport Schiphol
- CD database
- Used Audio Gear Database
- Musical note-sheets
- Bagism - A John Lennon fan page

- US Folk art broker
- Mail reading on the web
- Free home pages on www.somecoolname.mypage.org
- Der Server für Schulen im Web (In German)
- Auldhaefen Online Services
- CaryNET Information Center
- Dataden Computer Systems
- Andrémuseet (In Swedish)
- HOMESITE Internet Marketing
- Jade-V Network Services
- Weather World 2010 Technical Credits

- About The Gimp plugin registry
- Java tool Archiver technical detail (Slightly optimistic about **MySQL** ANSI-92 compliance)
- Games Domain Cheats Database
- The "Powered By" Page (Kcilink)
- Netcasting
- NBL (Australian National Basketball League) tipping
- CGI shop
- Whirlycott: Website Design
- Museum Tusculanum Press
- Centro Siciliano di Documentazione
- Quake statistics database
- Astroforum: Astrologie and related things (in German)
- OpenDebate - Interactive Polls & Open Discussion
- Online chemical dissertation server
- FreSch! The Free Scholarship Search Service
- Stockholm Pinball Locator
- HEK A construction company
- Elsevier Bussines Information
- Medical Links (Using Coldfusion and **MySQL**)
- Search for jobs & people at JobLink-USA
- Daily news about Linux in German language
- Competition Formation Skydiving
- E-commerce and internal accounting
- Denmark's leading business daily newspaper Børsen
- The Internet NES Database

- Travel agency in Prague in 3 languages
- Linkstation
- Searchable online database at Peoplestaff
- A searchable database system for horse classified ads
- The Poot site
- "Playin' in the LAN"; a network monitoring suite
- U.S. Army Publishing Agency
- Realestate handling in Yugoslavia
- PIMS; a Patient Information Management System
- Pilkington Software Inc
- Betazine - The Ultimate Online Beta Tester's Magazine
- A Vietnam Veteran's Memorial (The Wall) database.
- Gamer's Union specializes inauctions of used & out of print gaming material
- A daily bulletin at Monterey High school
- Computer Currents Magazine
- Community-owned site serving Lake Washington's Eastside residents and businesses
- French bowling site.

Send any additions to this list to `webmaster@mysql.com`.

Appendix B Contributed programs

Many users of **MySQL** have contributed *very* useful support tools and addons.

A list of what is available at `http://www.mysql.com/Contrib` (or any mirror) is shown below. If you want to build **MySQL** support for the Perl `DBI/DBD` interface, you should fetch the `Data-Dumper`, `DBI`, and `Msql-Mysql-modules` files and install them. See Section 4.10 [Perl support], page 48.

00-README This listing.

B.1 API's

- Perl modules
  - Data-Dumper-2.09.tar.gz Perl `Data-Dumper` module. Useful with `DBI/DBD` support.
  - DBI-1.11.tar.gz Perl `DBI` module.
  - KAMXbase1.0.tar.gz Convert between '`.dbf`' files and **MySQL** tables. Perl module written by Pratap Pereira `pereira@ee.eng.ohio-state.edu`, extened by Kevin A. McGrail `kmcgrail@digital1.peregrinehw.com`. This converter can handle MEMO fields.
  - Msql-Mysql-modules-1.2206.tar.gz Perl `DBD` module to access mSQL and **MySQL** databases..
  - Data-ShowTable-3.3.tar.gz Perl `Data-ShowTable` module. Useful with `DBI/DBD` support.
- JDBC
  - mm.mysql.jdbc-1.0.tar.gz New java JDBC driver for **MySQL**. This is a production release and is actively developed. By Mark Matthews (`mmatthew@ecn.purdue.edu`). This driver has a LGPL license. You can always find the newest driver at `http://www.worldserver.com/mm.mysql/`.
  - twz1jdbcForMysql-1.0.4-GA.tar.gz The twz driver: A type 4 JDBC driver by Terrence W. Zellers `zellert@voicenet.com`. This is commercial but is free for private and educational use.
- mysql-c++-0.02.tar.gz **MySQL** C++ wrapper library. By Roland Haenel, `rh@ginster.net`.
- mysql++.1.0.tar.gz **MySQL** C++ API (More than just a wrapper library). Originally by `kevina@clark.net`. Nowadays maintained by TCX.
- mysql-ruby-2.1.6.tar.gz **MySQL** Ruby module. By TOMITA Masahiro `tommy@tmtm.org` Ruby is Object-Oriented Interpreter Language.

  MySQL C++ API (More than a wrapper library). Originally by `kevina@clark.net`. Nowadays maintained by TCX.
- delphi-interface.gz Delphi interface to `libmysql.dll`, by Blestan Tabakov, `root@tdg.bis.bg`.
- DelphiMySQL2.zip Delphi interface to `libmysql.dll`, by `bsilva@umesd.k12.or.us`

- JdmMysqlDriver-0.1.0.tar.gz A VisualWorks 3.0 Smalltalk driver for **MySQL**. By joshmiller@earthlink.net
- Db.py Python module with caching. By gandalf@rosmail.com.
- MySQLmodule-1.4.tar.gz Python interface for the **MySQL**. By Joseph Skinner joe@earthlight.co.nz; Modified by Joerg Senekowitsch senekow@ibm.net
- mysql_mex.tar.gz An interface program for the Matlab program by MathWorks.
- mysqltcl-1.53.tar.gz Tcl interface for **MySQL**. Based on 'msqltcl-1.50.tar.gz'. Updated by Tobias Ritzau, tobri@ida.liu.se.
- MyC-0.1.tar.gz A Visual Basic-like API, by Ed Carp.
- sqlscreens-1.0.1.tar.gz TCL/TK code to generate database screens. By Jean-Francois Dockes.
- Vdb-dflts-2.1.tar.gz This is a new version of a set of library utilities intended to provide a generic interface to SQL database engines such that your application becomes a 3-tiered application. The advantage is that you can easily switch between and move to other database engines by implementing one file for the new backend without needing to make any changes to your applications. By damian@cablenet.net.
- DbFramework-1.10.tar.gz DbFramework is a collection of classes for manipulating **MySQL** databases. The classes are loosely based on the CDIF Data Model Subject Area. By Paul Sharpe paul@miraclefish.com.
- pike-mysql-1.4.tar.gz **MySQL** module for pike. For use with the Roxen web server.
- squile.tar.gz Module for `guile` that allows `guile` to interact with SQL databases. By Hal Roberts.
- stk-mysql.tar.gz Interface for Stk. Stk is the Tk widgets with Scheme underneath instead of Tcl. By Terry Jones
- eiffel-wrapper-1.0.tar.gz. Eiffel wrapper by Michael Ravits.

B.2 Clients

- Graphical clients
  - kmysqladmin-0.32.tar.gz
  - kmysqladmin-0.3-1.src.rpm
  - kmysqladmin-0.3-1.i386.rpm An administration tool for the **MySQL** server using QT / KDE. Tested only on Linux.
  - Java client using Swing By Fredy Fischer, se-afs@dial.eunet.ch. Ypu can always find the latest version here.
  - mysqlwinadmn.zip Win32 GUI (binary only) to administrate a database, by David B. Mansel, david@zhadum.org.
  - xmysqladmin-1.0.tar.gz A front end to the **MySQL** database engine. It allows reloads, status check, process control, isamchk, grant/revoke privileges, creating databases, dropping databases, create, alter, browse and drop tables. Originally by Gilbert Therrien, gilbert@ican.net but now in public domain and supported by TcX.

- xmysql-1.9.tar.gz
- xmysql home page A front end to the **MySQL** database engine. It allows for simple queries and table maintenance, as well as batch queries. By Rick Mehalick, dblhack@wt.net. Requires xforms 0.88 to work.

- Web clients
  - mysqladmin-atif-1.0.tar.gz WWW **MySQL** administrator for the `user`, `db` and `host` tables. By Tim Sailer, modified by Atif Ghaffar aghaffar@artemedia.ch.
  - mysql-webadmin-1.0a8-rz.tar.gz A tool written in PHP-FI to administrate **MySQL** databases remotely over the web within a Web-Browser. By Peter Kuppelwieser, peter.kuppelwieser@kantea.it. Updated by Wim Bonis, bonis@kiss.de. Not maintained anymore!
  - mysqladm.tar.gz **MySQL** Web Database Administration written in Perl. By Tim Sailer.
  - mysqladm-2.tar.gz Updated version of '`mysqladm.tar.gz`', by High Tide.
  - myadmin-0.4.tar.gz
  - MyAdmin home page A web based mysql administrator by Mike Machado.
  - phpMyAdmin_2.0.1.tar.gz A set of PHP3-scripts to adminstrate MySQL over the WWW.
  - phpMyAdmin home page A PHP3 tool in the spirit of mysql-webadmin, by Tobias Ratschiller, tobias@dnet.it
  - useradm.tar.gz **MySQL** administrator in PHP. By Ofni Thomas othomas@vaidsystems.com.

B.3 Web tools

- http://www.odbsoft.com/cook/sources.htm This package has various functions for generating html code from an SQL table structure and for generating SQL statements (Select, Insert, Update, Delete) from an html form. You can build a complete forms interface to an SQL database (query, add, update, delete) without any programming! By Marc Beneteau, marc@odbsoft.com.
- sqlhtml.tar.gz SQL/HTML is an HTML database manager for **MySQL** using `DBI` 1.06.
- UdmSearch-2.1.tar.gz A **MySQL**- and PHP- based search engine over http. By Alexander I. Barkov bar@izhcom.ru.
- wmtcl.doc
- wmtcl.lex With this you can write HTML files with inclusions of TCL code. By vvs@scil.npi.msu.su.
- www-sql-0.5.7.lsm
- www-sql-0.5.7.tar.gz A CGI program that parses an HTML file containing special tags, parses them and inserts data from a **MySQL** database.
- genquery.zip Perl SQL database interface package for html.
- cgi++-0.2.tar.gz A CGI wrapper by Sasha Pachev.

- WebBoard 1.0 EU-Industries Internet-Message-Board.
- DBIx-TextIndex-0.01.tar.gz Full-text searching with Perl on `BLOB/TEXT` columns by Daniel Koch.

B.4 Authentication tools

- ascend-radius-mysql-0.4.3.patch.gz This is authentication and logging patch using **MySQL** for Ascend-Radius. By `takeshi@SoftAgency.co.jp`.
- checkpassword-0.76-mysql-0.3.2.patch.gz MySQL authentication patch for QMAIL and checkpassword. These are useful for management user(mail,pop account) by **MySQL**. By `takeshi@SoftAgency.co.jp`
- jradius-diff.gz **MySQL** support for Livingston's Radius 2.01. Authentication and Accounting. By Jose de Leon, `jdl@thevision.net`
- mod_auth_mysql-2.20.tar.gz Apache authentication module for **MySQL**. By Zeev Suraski, `bourbon@netvision.net.il`.

  Please register this module at: `http://bourbon.netvision.net.il/mysql/mod_auth_mysql/register.html`. The registering information is only used for statistical purposes and will encourage further development of this module!
- mod_log_mysql-1.05.tar.gz **MySQL** logging module for Apache. By Zeev Suraski, `bourbon@netvision.net.il`.
- mypasswd-2.0.tar.gz Extra for `mod_auth_mysql`. This is a little tool that allows you to add/change user records storing group and/or password entries in **MySQL** tables. By Harry Brueckner, `brueckner@respublica.de`.
- mysql-passwd.README
- mysql-passwd-1.2.tar.gz Extra for `mod_auth_mysql`. This is a two-part system for use with `mod_auth_mysql`.
- pam_mysql.tar.gz This module authenticates users via `pam`, using **MySQL**.
- nsapi_auth_mysql.tar Netscape Web Server API (NSAPI) functions to authenticate (BASIC) users against **MySQL** tables. By Yuan John Jiang.
- qmail-1.03-mysql-0.3.2.patch.gz Patch for qmail to authenticate users from a **MySQL** table.
- pwcheck_mysql-0.1.tar.gz An authentication module for the Cyrus IMAP server. By Aaron Newsome.

B.5 Converters

- dbf2mysql-1.13.tgz Convert between '`.dbf`' files and **MySQL** tables. By Maarten Boekhold, `boekhold@cindy.et.tudelft.nl`, and Michael Widenius. This converter can't handle MEMO fields.
- dbf2mysql.zip Convert between FoxPro '`.dbf`' files and **MySQL** tables on Win32. By Alexander Eltsyn, `ae@nica.ru` or `ae@usa.net`.
- dump2h-1.20.gz Convert from `mysqldump` output to a C header file. By Harry Brueckner, `brueckner@mail.respublica.de`.

- exportsql.txt A script that is similar to `access_to_mysql.txt`, except that this one is fully configurable, has better type conversion (including detection of `TIMESTAMP` fields), provides warnings and suggestions while converting, quotes **all** special characters in text and binary data, and so on. It will also convert to `mSQL` v1 and v2, and is free of charge for anyone. See `http://www.cynergi.net/prod/exportsql/` for latest version. By Pedro Freire, `support@cynergi.net`. Note: Doesn't work with Access2!
- access_to_mysql.txt Paste this function into an Access module of a database which has the tables you want to export. See also `exportsql`. By Brian Andrews. Note: Doesn't work with Access2!
- importsql.txt A script that does the exact reverse of `exportsql.txt`. That is, it imports data from **MySQL** into an Access database via ODBC. This is very handy when combined with exportSQL, since it lets you use Access for all DB design and administration, and synchronize with your actual **MySQL** server either way. Free of charge. See `http://www.netdive.com/freebies/importsql/` for any updates. Created by Laurent Bossavit of NetDIVE. **Note:** Doesn't work with Access2!
- /msql2mysqlWrapper 1.0 A C wrapper from `mSQL` to **MySQL**. By `alfred@sb.net`
- sqlconv.pl A simple script that can be used to copy fields from one **MySQL** table to another in bulk. Basically, you can run `mysqldump` and pipe it to the `sqlconv.pl` script and the script will parse through the `mysqldump` output and will rearrange the fields so they can be inserted into a new table. An example is when you want to create a new table for a different site you are working on, but the table is just a bit different (ie - fields in different order, etc.). By Steve Shreeve.

B.6 Using MySQL with other products

- emacs-sql-mode.tar.gz Raw port of a SQL mode for XEmacs. Supports completion. Original by Peter D. Pezaris `pez@atlantic2.sbi.com` and partial **MySQL** port by David Axmark.
- MyAccess.mda MyAccess is an AddIn for Access 97 and handles a lot of maintanance work for **MySQL** databases. MyAccess-readme. By Hubertus Hiden.
- radius-0.3.tar.gz Patches for `radiusd` to make it support **MySQL**. By Wim Bonis, `bonis@kiss.de`.

B.7 Useful tools

- mysql_watchdog.pl Monitor the **MySQL** daemon for possible lockups. By Yermo Lamers, `yml@yml.com`.
- mysqltop.tar.gz Sends a query in a fixed time interval to the server and shows the resulting table. By Thomas Wana.
- mysql_structure_dumper.tar.gz Prints out the structure of the all tables in a database. By Thomas Wana.
- structure_dumper.tgz Prints the structure of every table in a database. By Thomas Wana.

- mysqlsync-1.0-alpha.tar.gz. A perl script to keep remote copies of a **MySQL** database in sync with a central master copy. By Mark Jeftovic. markjr@easydns.com
- MySQLTutor. MySQLTutor. A tutor of **MySQL** for beginners</A>
- MySQLDB.zip A COM library for **MySQL** by Alok Singh.
- MySQLDB-readme.html
- mysql_replicate.pl Perl program that handles replication. By mjs@atnet.net.au
- DBIx-TextIndex-0.02.tar.gz Perl program that uses reverse indexing to handle text searching. By Daniel Koch.

B.8 Uncategorized

- findres.pl Find reserved words in tables. By Nem W Schlecht.
- handicap.tar.gz Performance handicapping system for yachts. Uses PHP. By rhill@stobyn.ml.org.
- hylalog-1.0.tar.gz Store `hylafax` outgoing faxes in a **MySQL** database. By Sinisa Milivojevic, sinisa@coresinc.com.
- mrtg-mysql-1.0.tar.gz **MySQL** status plotting with MRTG, by Luuk de Boer, luuk@wxs.nl.
- wuftpd-2.4.2.18-mysql_support.2.tar.gz Patches to add logging to **MySQL** for WU-ftpd. By Zeev Suraski, bourbon@netvision.net.il.
- Old-Versions Previous versions of things found here that you probably won't be interested in.

Appendix C Contributors to MySQL

Contributors to the **MySQL** distribution are listed below, in somewhat random order:

Michael (Monty) Widenius
: Has written the following parts of **MySQL**:

  - All the main code in `mysqld`.
  - New functions for the string library.
  - Most of the `mysys` library.
  - The `NISAM` and `MyISAM` libraries (B-tree index file handlers with index compression and different record formats).
  - The `heap` library. A memory table system with our superior full dynamic hashing. In use since 1981 and published around 1984.
  - The `replace` program (look into it, it's COOL!).
  - **MyODBC**, the ODBC driver for Windows95.
  - Fixing bugs in MIT-pthreads to get it to work for **MySQL**. And also Unireg, a curses-based application tool with many utilities.
  - Porting of `mSQL` tools like `msqlperl`, `DBD/DBI` and `DB2mysql`.
  - Most parts of crash-me and the **MySQL** benchmarks.

David Axmark
  - Coordinator and main writer for the **Reference Manual**, including enhancements to `texi2html`. Also automatic website updating from this manual.
  - Autoconf, Automake and `libtool` support.
  - The licensing stuff.
  - Parts of all the text files. (Nowadays only the '`README`' is left. The rest ended up in the manual.)
  - Our Mail master.
  - Lots of testing of new features.
  - Our in-house "free" software lawyer.
  - Mailing list maintainer (who never has the time to do it right...)
  - Our original portability code (more than 10 years old now). Nowadays only some parts of `mysys` are left.
  - Someone for Monty to call in the middle of the night when he just got that new feature to work. :-)

Paul DuBois
: Help with making the Reference Manual correct and understandable.

Gianmassimo Vigazzola `qwerg@mbox.vol.it` or `qwerg@tin.it`
: The initial port to Win32/NT.

Kim Aldale
: Rewriting Monty's and David's attempts at English into English.

Allan Larsson (The BOSS at TcX)
: For all the time he has allowed Monty to spend on this "maybe useful" tool (**MySQL**). Dedicated user (and bug finder) of Unireg & **MySQL**.

Per Eric Olsson
: For more or less constructive criticism and real testing of the dynamic record format.

Irena Pancirov irena@mail.yacc.it
: Win32 port with Borland compiler.

David J. Hughes
: For the effort to make a shareware SQL database. We at TcX started with `mSQL`, but found that it couldn't satisfy our purposes so instead we wrote a SQL interface to our application builder Unireg. `mysqladmin` and `mysql` are programs that were largely influenced by their `mSQL` counterparts. We have put a lot of effort into making the **MySQL** syntax a superset of `mSQL`. Many of the APIs ideas are borrowed from `mSQL` to make it easy to port free `mSQL` programs to **MySQL**. **MySQL** doesn't contain any code from `mSQL`. Two files in the distribution ('`client/insert_test.c`' and '`client/select_test.c`') are based on the corresponding (non-copyrighted) files in the `mSQL` distribution, but are modified as examples showing the changes necessary to convert code from `mSQL` to **MySQL**. (`mSQL` is copyrighted David J. Hughes.)

Fred Fish
: For his excellent C debugging and trace library. Monty has made a number of smaller improvements to the library (speed and additional options).

Richard A. O'Keefe
: For his public domain string library.

Henry Spencer
: For his regex library, used in `WHERE column REGEXP regexp`.

Free Software Foundation
: From whom we got an excellent compiler (`gcc`), the `libc` library (from which we have borrowed '`strto.c`' to get some code working in Linux) and the `readline` library (for the `mysql` client).

Free Software Foundation & The XEmacs development team
: For a really great editor/environment used by almost everybody at TcX/detron.

Igor Romanenko igor@frog.kiev.ua
: `mysqldump` (previously `msqldump`, but ported and enhanced by Monty).

Tim Bunce, Alligator Descartes
: For the `DBD` (Perl) interface.

Andreas Koenig a.koenig@mind.de
: For the Perl interface to **MySQL**.

Eugene Chan eugene@acenet.com.sg
: For porting PHP to **MySQL**.

Michael J. Miller Jr. mke@terrapin.turbolift.com
: For the first **MySQL** manual. And a lot of spelling/language fixes for the FAQ (that turned into the **MySQL** manual a long time ago).

Giovanni Maruzzelli maruzz@matrice.it
: For porting iODBC (Unix ODBC).

Chris Provenzano
: Portable user level pthreads. From the copyright: This product includes software developed by Chris Provenzano, the University of California, Berkeley, and contributors. We are currently using version 1_60_beta6 patched by Monty (see '`mit-pthreads/Changes-mysql`').

Xavier Leroy Xavier.Leroy@inria.fr
: The author of LinuxThreads (used by **MySQL** on Linux).

Zarko Mocnik zarko.mocnik@dem.si
: Sorting for Slovenian language and the '`cset.tar.gz`' module that makes it easier to add other character sets.

"TAMITO" tommy@valley.ne.jp
: The `_MB` character set macros and the ujis and sjis character sets.

Yves Carlier Yves.Carlier@rug.ac.be
: `mysqlaccess`, a program to show the access rights for a user.

Rhys Jones rhys@wales.com (And GWE Technologies Limited)
: For the JDBC, a module to extract data from **MySQL** with a Java client.

Dr Xiaokun Kelvin ZHU X.Zhu@brad.ac.uk
: Further development of the JDBC driver and other **MySQL**-related Java tools.

James Cooper pixel@organic.com
: For setting up a searchable mailing list archive at his site.

Rick Mehalick Rick_Mehalick@i-o.com
: For `xmysql`, a graphical X client for **MySQL**.

Doug Sisk sisk@wix.com
: For providing RPM packages of **MySQL** for RedHat Linux.

Diemand Alexander V. axeld@vial.ethz.ch
: For providing RPM packages of **MySQL** for RedHat Linux/Alpha.

Antoni Pamies Olive toni@readysoft.es
: For providing RPM versions of a lot of **MySQL** clients for Intel and SPARC.

Jay Bloodworth jay@pathways.sde.state.sc.us
: For providing RPM versions for **MySQL** 3.21 versions.

Jochen Wiedmann wiedmann@neckar-alb.de
: For maintaining the Perl `DBD::mysql` module.

Therrien Gilbert gilbert@ican.net, Jean-Marc Pouyot jmp@scalaire.fr
: French error messages.

Petr snajdr, snajdr@pvt.net
: Czech error messages.

Jaroslaw Lewandowski jotel@itnet.com.pl
: Polish error messages.

Miguel Angel Fernandez Roiz
: Spanish error messages.

Roy-Magne Mo rmo@www.hivolda.no
: Norwegian error messages and testing of 3.21.#.

Timur I. Bakeyev root@timur.tatarstan.ru
: Russian error messages.

brenno@dewinter.com && Filippo Grassilli phil@hyppo.com
: Italian error messages.

Dirk Munzinger dirk@trinity.saar.de
: German error messages.

Billik Stefan billik@sun.uniag.sk
: Slovak error messages.

David Sacerdote davids@secnet.com
: Ideas for secure checking of DNS hostnames.

Wei-Jou Chen jou@nematic.ieo.nctu.edu.tw
: Some support for Chinese(BIG5) characters.

Zeev Suraski bourbon@netvision.net.il
: `FROM_UNIXTIME()` time formatting, `ENCRYPT()` functions, and `bison` adviser. Active mailing list member.

Luuk de Boer luuk@wxs.nl
: Ported (and extended) the benchmark suite to `DBI/DBD`. Have been of great help with `crash-me` and running benchmarks. Some new date functions. The mysql_setpermissions script.

Jay Flaherty fty@utk.edu
: Big parts of the Perl `DBI/DBD` section in the manual.

Paul Southworth pauls@etext.org, Ray Loyzaga yar@cs.su.oz.au
: Proof-reading of the Reference Manual.

Alexis Mikhailov root@medinf.chuvashia.su
: User definable functions (UDFs); `CREATE FUNCTION` and `DROP FUNCTION`.

Andreas F. Bobak bobak@relog.ch
: The `AGGREGATE` extension to UDF functions.

Ross Wakelin R.Wakelin@march.co.uk
: Help to set up InstallShield for MySQL-Win32.

Jethro Wright III jetman@li.net
: The '`libmysql.dll`' library.

James Pereria jpereira@iafrica.com
: Mysqlmanager, a Win32 GUI tool for administrating **MySQL**.

Curt Sampson cjs@portal.ca
: Porting of MIT-pthreads to NetBSD/Alpha and NetBSD 1.3/i386.

Sinisa Milivojevic sinisa@coresinc.com
: Compression (with `zlib`) to the client/server protocol. Perfect hashing for the lexical analyzer phase.

Antony T. Curtis antony.curtis@olcs.net
: Porting of **MySQL** to OS/2.

Martin Ramsch m.ramsch@computer.org
: Examples in the **MySQL** Tutorial.

Other contributors, bugfinders and testers: James H. Thompson, Maurizio Menghini, Wojciech Tryc, Luca Berra, Zarko Mocnik, Wim Bonis, Elmar Haneke, jehamby@lightside, psmith@BayNetworks.COM, duane@connect.com.au, Ted Deppner ted@psyber.com, Mike Simons, Jaakko Hyvätti.

And lots of bug report/patches from the folks on the mailing list.

And a big tribute to those that help us answer questions on the `mysql@lists.mysql.com` mailing list:

Daniel Koch dkoch@amcity.com
: Irix setup.

Luuk de Boer luuk@wxs.nl
: Benchmark questions.

Tim Sailer tps@users.buoy.com
: `DBD-mysql` questions.

Boyd Lynn Gerber gerberb@zenez.com
: SCO related questions.

Richard Mehalick RM186061@shellus.com
: `xmysql`-releated questions and basic installation questions.

Zeev Suraski bourbon@netvision.net.il
: Apache module configuration questions (log & auth), PHP-related questions, SQL syntax related questions and other general questions.

Francesc Guasch frankie@citel.upc.es
: General questions.

Jonathan J Smith jsmith@wtp.net
: Questions pertaining to OS-specifics with Linux, SQL syntax, and other things that might be needing some work.

David Sklar sklar@student.net
: Using **MySQL** from PHP and Perl.

Alistair MacDonald A.MacDonald@uel.ac.uk
: Not yet specified, but is flexible and can handle Linux and maybe HP-UX. Will try to get user to use `mysqlbug`.

John Lyon jlyon@imag.net
: Questions about installing **MySQL** on Linux systems, using either '`.rpm`' files, or compiling from source.

Lorvid Ltd. lorvid@WOLFENET.com
: Simple billing/license/support/copyright issues.

Patrick Sherrill patrick@coconet.com
: ODBC and VisualC++ interface questions.

Randy Harmon rjharmon@uptimecomputers.com
: `DBD`, Linux, some SQL syntax questions.

Appendix D MySQL change history

Note that we tend to update the manual at the same time we implement new things to **MySQL**. If you find a version listed below that you can't find on the **MySQL** download page, this means that the version has not yet been released!

D.1 Changes in release 3.23.x (Released as alpha)

The major difference between release 3.23 and releases 3.22 and 3.21 is that 3.23 contains a new ISAM library (MyISAM), which is more tuned for SQL than the old ISAM was.

The 3.23 release is under development, and things will be added at a fast pace to it. For the moment we recommend this version only for users that desperately need a new feature that is found only in this release (like big file support and machine-independent tables). (Note that all new functionality in MySQL 3.23 is extensively tested, but as this release involves much new code, it's difficult to test everything). This version should start to stabilise as soon as we get subselects included in it.

D.1.1 Changes in release 3.23.3

Added patches for mit-pthreads on NetBSD

- Fixed range bug in MyISAM.
- `ASC` is now again default for `ORDER BY`.
- Added `LIMIT` to `UPDATE`.
- New client function: `mysql_change_user`.
- Added character set to `SHOW VARIABLES`
- Added support of `--[whitespace]` comments.
- Allow `INSERT into table_name VALUES ()`'
- Changed `SUBSTRING(text FROM pos)` to conform to ANSI SQL. (Before this construct returned the rightmost 'pos' characters).
- `SUM(..)` with `GROUP BY` returned 0 on some system.
- Changed output for `SHOW TABLE STATUS`.
- Added `DELAY_KEY_WRITE` option to `CREATE TABLE`.
- Allow `AUTO_INCREMENT` on any key part.
- Fixed problem with `YEAR(NOW())` and `YEAR(CURDATE())`.
- Added `CASE` construct.
- New function `COALESCE()`.

D.1.2 Changes in release 3.23.2

- Fixed range optimizer bug: `SELECT * FROM table_name WHERE key_part1 >= const AND (key_part2 = const OR key_part2 = const)`. The bug was that some rows could be duplicated in the result.

- Running `myisamchk` without `-a` updated the index distribution wrong.
- `SET SQL_LOW_PRIORITY_UPDATES=1` gave parse error before.
- You can now update indexes columns that are used in the `WHERE` clause. `UPDATE tbl_name SET KEY=KEY+1 WHERE KEY > 100`
- Date handling should now be a bit faster.
- Added handling of fuzzy dates (dates where day or month is 0): (Like: 1999-01-00)
- Fixed optimization of `SELECT ... WHERE key_part1=const1 AND key_part_2=const2 AND key_part1=const4 AND key_part2=const4` ; Indextype should be `range` instead of `ref`.
- Fixed `egcs` 1.1.2 optimizer bug (when using `BLOB`s) on Linux Alpha.
- Fixed problem with `LOCK TABLES` combined with `DELETE FROM table`.
- MyISAM tables now allow keys on `NULL` and `BLOB/TEXT` columns.
- The following join is now much faster: `SELECT ... FROM t1 LEFT JOIN t2 ON ... WHERE t2.not_null_column IS NULL`.
- `ORDER BY` and `GROUP BY` can be done on functions.
- Changed handling of 'const_item' to allow handling of `ORDER BY RAND()`.
- Indexes are now used for `WHERE key_column = function`.
- Indexes are now used for `WHERE key_column = column_name` even if the columns are not identically packed.
- Indexes are now used for `WHERE column_name IS NULL`.
- Changed heap tables to be stored in low_byte_first order (to make it easy to convert to MyISAM tables)
- Automatic change of HEAP temporary tables to MyISAM tables in case of 'table is full' errors.
- Added option `--init-file=file_name` to `mysqld`.
- `COUNT(DISTINCT value,[value,...])`
- `CREATE TEMPORARY TABLE` now creates a temporary table, in its own namespace, that is automatically deleted if connection is dropped.
- New keywords (required for `CASE`: `CASE, THEN, WHEN, ELSE and END`.
- New functions `EXPORT_SET()` and `MD5()`.
- Support for the GB2312 Chinese character set.

D.1.3 Changes in release 3.23.1

- Fixed some compilation problems.

D.1.4 Changes in release 3.23.0

A new table handler library (`MyISAM`) with a lot of new features. See Section 10.18 [Table types], page 262.

- You can create in-memory `HEAP` tables which are extremely fast for lookups.

- Support for big files (63 bit) on OSes that support big files.
- New function `LOAD_FILE(filename)` to get the contents of a file as a string value.
- New operator `<=>` which will act as `=` but will return TRUE if both arguments are `NULL`. This is useful for comparing changes between tables.
- Added the ODBC 3.0 `EXTRACT(interval FROM datetime)` function.
- Columns defined as `FLOAT(4)` or `FLOAT(8)` are not rounded on storage and may be in scientific notation (1.0 E+10) when retrieved.
- `REPLACE` is now faster than before.
- Changed `LIKE` character comparison to behave as `=`; This means that `'e' LIKE ''` is now true.
- `SHOW TABLE STATUS` returns a lot of information about the tables.
- Added `LIKE` to the `SHOW STATUS` command.
- Added privilege column to `SHOW COLUMNS`.
- Added columns `packed` and `comment` to `SHOW INDEX`.
- Added comments to tables (with `CREATE TABLE ... COMMENT "xxx"`).
- Added `UNIQUE`, as in `CREATE TABLE table_name (col int not null UNIQUE)`
- New create syntax: `CREATE TABLE SELECT ....`
- New create syntax: `CREATE TABLE IF NOT EXISTS ...`
- Allow creation of `CHAR(0)` columns.
- `DATE_FORMAT()` now requires `%` before any format character.
- `DELAYED` is now a reserved word (sorry about that :( ).
- An example procedure is added: `analyse`, file: '`sql_analyse.c`'. This will describe the data in your query. Try the following:

  ```
  SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max elements,[max memory]])
  ```

  This procedure is extremely useful when you want to check the data in your table!
- `BINARY` cast to force a string to be compared case sensitively.
- Added option `--skip-show-databases` to `mysqld`.
- Check if a row has changed in an `UPDATE` now also works with `BLOB`/`TEXT` columns.
- Added the `INNER` join syntax. **NOTE**: This made `INNER` an reserved word!
- Added support for netmasks to the hostname in the **MySQL** tables. You can specify a netmask using the `IP/NETMASK` syntax.
- If you compare a `NOT NULL DATE/DATETIME` column with `IS NULL`, this is changed to a compare against `0` to satisfy some ODBC applications. (By shreeve@uci.edu).
- `NULL IN (...)` now returns `NULL` instead of `0`. This will ensure that `null_column NOT IN (...)` doesn't match `NULL` values.
- Fix storage of floating point values in `TIME` columns.
- Changed parsing of `TIME` strings to be more strict. Now the fractional second part is detected (and currently skipped). The following formats are supported:

  ```
  [[DAYS] [H]H:]MM:]SS[.fraction]
  ```

`[[[[[H]H]H]H]MM]SS[.fraction]`

- Detect (and ignore) second fraction part from `DATETIME`.
- Added the `LOW_PRIORITY` attribute to `LOAD DATA INFILE`.
- The default index name now uses the same case as the used column name.
- Changed default number of connections to 100.
- Use bigger buffers when using `LOAD DATA INFILE`.
- `DECIMAL(x,y)` now works according to ANSI SQL.
- Added aggregate UDF functions. Thanks to Andreas F. Bobak `bobak@relog.ch` for this!
- `LAST_INSERT_ID()` is now updated for `INSERT INTO ... SELECT`.
- Some small changes to the join table optimizer to make some joins faster.
- `SELECT DISTINCT` is much faster; It uses the new `UNIQUE` functionality in `MyISAM`. One difference compared to **MySQL** 3.22 is that the output of `DISTINCT` is not sorted anymore.
- All C client API macros are now functions to make shared libraries more reliable. Because of this, you can no longer call `mysql_num_fields()` on a `MYSQL` object, you must use `mysql_field_count()` instead.
- Added use of `LIBEWRAP`; Patch by Henning P . Schmiedehausen.
- Don't allow `AUTO_INCREMENT` for other than numerical columns.
- Using `AUTO_INCREMENT` will now automatically make the column `NOT NULL`.
- Show `NULL` as the default value for AUTO_INCREMENT columns.
- Added `SQL_BIG_RESULT`; `SQL_SMALL_RESULT` is now default.
- Added a shared library RPM. This enchancement was contributed by David Fox (dsfox@cogsci.ucsd.edu).
- Added a `--enable-large-files/--disable-large-files` switch to `configure`. See '`configure.in`' for some systems where this is automatically turned off because of broken implementations.
- Upgraded `readline` to 4.0.
- New `CREATE TABLE` options: `PACK_KEYS` and `CHECKSUM`.
- Added `mysqld` option `--default-table-type`.

D.2 Changes in release 3.22.x

The 3.22 version has faster and safer connect code and a lot of new nice enhancements. The reason for not including these changes in the 3.21 version is mainly that we are trying to avoid big changes to 3.21 to keep it as stable as possible. As there aren't really any MAJOR changes, upgrading to 3.22 should be very easy and painless. See Section 4.16.2 [Upgrading-from-3.21], page 83.

3.22 should also be used with the new `DBD-mysql` (1.20xx) driver that can use the new connect protocol!

D.2.1 Changes in release 3.22.26

- Fixed core dump with empty `BLOB/TEXT` column to `REVERSE()`.
- Extended `/*! */` with version numbers.
- Changed `SUBSTRING(text FROM pos)` to conform to ANSI SQL. (Before this construct returned the rightmost 'pos' characters).
- Fixed problem with `LOCK TABLES` combined with `DELETE FROM table`
- Fixed problem that INSERT ... SELECT didn't use SQL_BIG_TABLES.
- `SET SQL_LOW_PRIORITY_UPDATES=#` didn't work.
- Password wasn't updated correctly if privileges didn't change on: `GRANT ... IDENTIFIED BY`
- Fixed range optimizer bug in `SELECT * FROM table_name WHERE key_part1 >= const AND (key_part2 = const OR key_part2 = const)`
- Fixed bug in compression key handling in NISAM.

D.2.2 Changes in release 3.22.25

- Fixed some small problems with the installation.

D.2.3 Changes in release 3.22.24

- `DATA` is not a reserved word anymore.
- Fixed optimizer bug with tables with only one row.
- Fixed bug when using `LOCK TABLES table_name READ; FLUSH TABLES;`
- Applied some patches for HP-UX.
- `isamchk` should now work on Win32.
- Changed '`configure`' to not use big file handling on Linux as this crashes some RedHat 6.0 systems

D.2.4 Changes in release 3.22.23

- Upgraded to use Autoconf 2.13, Automake 1.4 and `libtool` 1.3.2.
- Better support for SCO in `configure`.
- Added option `--defaults-file=###` to option file handling to force use of only one specific option file.
- Extended `CREATE` syntax to ignore `MySQL 3.23` keywords.
- Fixed deadlock problem when using `INSERT DELAYED` on a table locked with `LOCK TABLES`.
- Fixed deadlock problem when using `DROP TABLE` on a table that was locked by another thread.
- Add logging of `GRANT/REVOKE` commands in the update log.
- Fixed `isamchk` to detect a new error condition.
- Fixed bug in `NATURAL LEFT JOIN`.

D.2.5 Changes in release 3.22.22

- Fixed problem in the C API when you called `mysql_close()` directly after `mysql_init()`.
- Better client error message when you can't open socket.
- Fixed `delayed_insert_thread` counting when you couldn't create a new delayed_insert thread.
- Fixed bug in `CONCAT()` with many arguments.
- Added patches for DEC 3.2 and SCO.
- Fixed path-bug when installing **MySQL** as a service on NT.
- The **MySQL**-win32 version is now compiled with VC++ 6.0 instead of with VC++ 5.0.
- New installation setup for **MySQL-Win32**

D.2.6 Changes in release 3.22.21

- Fixed problem with `DELETE FROM TABLE` when table was locked by another thread.
- Fixed bug in `LEFT JOIN` involving empty tables.
- Changed the `mysql.db` column from `char(32)` to `char(60)`.
- `MODIFY` and `DELAYED` are not reserved words anymore.
- Fixed a bug when storing days in a `TIME` column.
- Fixed a problem with `Host '..' is not allowed to connect to this MySQL server` after one had inserted a new **MySQL** user with a `GRANT` command.
- Changed to use TCP_NODELAY also on Linux (Should give faster TCP/IP connections).

D.2.7 Changes in release 3.22.20

- Fixed `STD()` for big tables when result should be 0.
- The update log didn't have newlines on some operating systems.
- `INSERT DELAYED` had some garbage at end in the update log.

D.2.8 Changes in release 3.22.19

- Fixed bug in `mysql_install_db` (from 3.22.17).
- Changed default key cache size to 8M.
- Fixed problem with queries that needed temporary tables with `BLOB` columns.

D.2.9 Changes in release 3.22.18

- Fixes a fatal problem in 3.22.17 on Linux; After `shutdown` all threads didn't die properly.
- Added option `-O flush-time=#` to `mysqld`. This is mostly useful on Win32 and tells how often **MySQL** should close all unused tables and flush all updated tables to disk.
- Fixed problem that a `VARCHAR` column compared with `CHAR` column didn't use keys efficiently.

D.2.10 Changes in release 3.22.17

- Fixed a core dump problem when using `--log-update` and connecting without a default database.
- Fixed some `configure` and portability problems.
- Using `LEFT JOIN` on tables that had circular dependencies caused `mysqld` to hang forever.

D.2.11 Changes in release 3.22.16

- `mysqladmin processlist` could kill the server if a new user logged in.
- `DELETE FROM tbl_name WHERE key_column=col_name` didn't find any matching rows. Fixed.
- `DATE_ADD(column,...)` didn't work.
- `INSERT DELAYED` could deadlock with status 'upgrading lock'
- Extended `ENCRYPT()` to take longer salt strings than 2 characters.
- `longlong2str` is now much faster than before. For `Intel x86` platforms, this function is written in optimized assembler.
- Added the `MODIFY` keyword to `ALTER TABLE`.

D.2.12 Changes in release 3.22.15

- `GRANT` used with `IDENTIFIED BY` didn't take effect until privileges were flushed.
- Name change of some variables in `SHOW STATUS`.
- Fixed problem with `ORDER BY` with 'only index' optimzation when there were multiple key definitions for a used column.
- `DATE` and `DATETIME` columns are now up to 5 times faster than before.
- `INSERT DELAYED` can be used to let the client do other things while the server inserts rows into a table.
- `LEFT JOIN USING (col1,col2)` didn't work if one used it with tables from 2 different databases.
- `LOAD DATA LOCAL INFILE` didn't work in the Unix version because of a missing file.
- Fixed problems with `VARCHAR/BLOB` on very short rows (< 4 bytes); error 127 could occur when deleting rows.
- Updating `BLOB/TEXT` through formulas didn't work for short (< 256 char) strings.
- When you did a `GRANT` on a new host, `mysqld` could die on the first connect from this host.
- Fixed bug when one used `ORDER BY` on column name that was the same name as an alias.
- Added `BENCHMARK(loop-count,expression)` function to time expressions.

D.2.13 Changes in release 3.22.14

- Allow empty arguments to `mysqld` to make it easier to start from shell scripts.
- Setting a `TIMESTAMP` column to `NULL` didn't record the timestamp value in the update log.
- Fixed lock handler bug when one did `INSERT INTO TABLE ... SELECT ... GROUP BY`.
- Added a patch for `localtime_r()` on Win32 so that it will not crash anymore if your date is > 2039, but instead will return a time of all zero.
- Names for user-defined functions are no longer case sensitive.
- Added escape of `^Z` (ASCII 26) to `\Z` as `^Z` doesn't work with pipes on Win32.
- `mysql_fix_privileges` adds a new column to the `mysql.func` to support aggregate UDF functions in future **MySQL** releases.

D.2.14 Changes in release 3.22.13

- Saving `NOW()`, `CURDATE()` or `CURTIME()` directly in a column didn't work.
- `SELECT COUNT(*) ... LEFT JOIN ...` didn't work with no `WHERE` part.
- Updated '`config.guess`' to allow **MySQL** to configure on UnixWare 7.0.x.
- Changed the implementation of `pthread_cond()` on the Win32 version. `get_lock()` now correctly times out on Win32!

D.2.15 Changes in release 3.22.12

- Fixed problem when using `DATE_ADD()` and `DATE_SUB()` in a `WHERE` clause.
- You can now set the password for a user with the `GRANT ... TO user IDENTIFIED BY 'password'` syntax.
- Fixed bug in `GRANT` checking with `SELECT` on many tables.
- Added missing file `mysql_fix_privilege_tables` to the RPM distribution. This is not run by default since it relies on the client package.
- Added option `SQL_SMALL_RESULT` to `SELECT` to force use of fast temporary tables when you know that the result set will be small.
- Allow use of negative real numbers without a decimal point.
- Day number is now adjusted to max days in month if the resulting month after `DATE_ADD/DATE_SUB()` doesn't have enough days.
- Fix that `GRANT` compares columns in case-insensitive fashion.
- Fixed a bug in '`sql_list.h`' that made `ALTER TABLE` dump core in some contexts.
- The hostname in `user@hostname` can now include '`.`' and '`-`' without quotes in the context of the `GRANT`, `REVOKE` and `SET PASSWORD FOR ...` statements.
- Fix for `isamchk` for tables which need big temporary files.

D.2.16 Changes in release 3.22.11

- **IMPORTANT**: You must run the `mysql_fix_privilege_tables` script when you upgrade to this version! This is needed because of the new `GRANT` system. If you don't do this, you will get `Access denied` when you try to use `ALTER TABLE`, `CREATE INDEX` or `DROP INDEX`.
- `GRANT` to allow/deny users table and column access.
- Changed `USER()` to return `user@host`
- Changed the syntax for how to set `PASSWORD` for another user.
- New command `FLUSH STATUS` that sets most status variables to zero.
- New status variables: `aborted_threads`, `aborted_connects`.
- New option variable: `connection_timeout`.
- Added support for Thai sorting (by Pruet Boonma pruet@ds90.intanon.nectec.or.th).
- Slovak and japanese error messages.
- Configuration and portability fixes.
- Added option `SET SQL_WARNINGS=1` to get a warning count also for simple inserts.
- **MySQL** now uses `SIGTERM` instead of `SIGQUIT` with shutdown to work better on FreeBSD.
- Added option `\G` (print vertically) to `mysql`.
- `SELECT HIGH_PRIORITY` ... killed `mysqld`.
- `IS NULL` on a `AUTO_INCREMENT` column in a `LEFT JOIN` didn't work as expected.
- New function `MAKE_SET()`.

D.2.17 Changes in release 3.22.10

- `mysql_install_db` no longer starts the **MySQL** server! You should start `mysqld` with `safe_mysqld` after installing it! The **MySQL** RPM will however start the server as before.
- Added `--bootstrap` option to `mysqld` and recoded `mysql_install_db` to use it. This will make it easier to install **MySQL** with RPMs.
- Changed `+`, `-` (sign and minus), `*`, `/`, `%`, `ABS()` and `MOD()` to be `BIGINT` aware (64-bit safe).
- Fixed a bug in `ALTER TABLE` that caused `mysqld` to crash.
- **MySQL** now always reports the conflicting key values when a duplicate key entry occurs. (Before this was only reported for `INSERT`).
- New syntax: `INSERT INTO tbl_name SET col_name=value,col_name=value,...`
- Most errors in the '`.err`' log are now prefixed with a time stamp.
- Added option `MYSQL_INIT_COMMAND` to `mysql_options()` to make a query on connect or reconnect.
- Added option `MYSQL_READ_DEFAULT_FILE` and `MYSQL_READ_DEFAULT_GROUP` to `mysql_options()` to read the following parameters from the **MySQL** option files: port, socket, compress, password, pipe, timeout, user, init-command, host and database.

- Added `maybe_null` to the UDF structure.
- Added option `IGNORE` to `INSERT` statemants with many rows.
- Fixed some problems with sorting of the koi8 character sets; Users of koi8 **MUST** run `isamchk -rq` on each table that has an index on a `CHAR` or `VARCHAR` column.
- New script `mysql_setpermission`, by Luuk de Boer, allows one to easily create new users with permissions for specific databases.
- Allow use of hexadecimal strings (0x...) when specifying a constant string (like in the column separators with `LOAD DATA INFILE`).
- Ported to OS/2 (thanks to Antony T. Curtis antony.curtis@olcs.net).
- Added more variables to `SHOW STATUS` and changed format of output to be like `SHOW VARIABLES`.
- Added `extended-status` command to `mysqladmin` which will show the new status variables.

D.2.18 Changes in release 3.22.9

- `SET SQL_LOG_UPDATE=0` caused a lockup of the server.
- New SQL command: `FLUSH [ TABLES | HOSTS | LOGS | PRIVILEGES ] [, ...]`
- New SQL command: `KILL` thread_id
- Added casts and changed include files to make **MySQL** easier to compile on AIX and DEC OSF1 4.x
- Fixed conversion problem when using `ALTER TABLE` from a `INT` to a short `CHAR()` column.
- Added `SELECT HIGH_PRIORITY`; This will get a lock for the `SELECT` even if there is a thread waiting for another `SELECT` to get a `WRITE LOCK`.
- Moved wild_compare to string class to be able to use `LIKE` on `BLOB/TEXT` columns with `\0`.
- Added `ESCAPE` option to `LIKE`.
- Added a lot more output to `mysqladmin debug`.
- You can now start `mysqld` on Win32 with the `--flush` option. This will flush all tables to disk after each update. This makes things much safer on NT/Win98 but also **MUCH** slower.

D.2.19 Changes in release 3.22.8

- Czech character sets should now work much better. You must also install ftp://www.mysql.com/pub/mysql/Downloads/Patches/czech-3.22.8-patch. This patch should also be installed if you are using a character set with uses `my_strcoll()`! The patch should always be safe to install (for any system), but as this patch changes ISAM internals it's not yet in the default distribution.
- `DATE_ADD()` and `DATE_SUB()` didn't work with group functions.
- `mysql` will now also try to reconnect on `USE DATABASE` commands.

- Fix problem with `ORDER BY` and `LEFT JOIN` and `const` tables.
- Fixed problem with `ORDER BY` if the first `ORDER BY` column was a key and the rest of the `ORDER BY` columns wasn't part of the key.
- Fixed a big problem with `OPTIMIZE TABLE`.
- **MySQL** clients on NT will now by default first try to connect with named pipes and after this with TCP/IP.
- Fixed a problem with `DROP TABLE` and `mysqladmin shutdown` on Win32 (a fatal bug from 3.22.6).
- Fixed problems with `TIME columns` and negative strings.
- Added an extra thread signal loop on shutdown to avoid some error messages from the client.
- **MySQL** now uses the next available number as extension for the update log file.
- Added patches for UNIXWARE 7.

D.2.20 Changes in release 3.22.7

- Added `LIMIT` clause for the `DELETE` statement.
- You can now use the `/*! ... */` syntax to hide **MySQL**-specific keywords when you write portable code. **MySQL** will parse the code inside the comments as if the surrounding `/*!` and `*/` comment characters didn't exist.
- `OPTIMIZE TABLE tbl_name` can now be used to reclaim disk space after many deletes. Currently, this uses `ALTER TABLE` to re-generate the table, but in the future it will use an integrated `isamchk` for more speed.
- Upgraded `libtool` to get the configure more portable.
- Fixed slow `UPDATE` and `DELETE` operations when using `DATETIME` or `DATE` keys.
- Changed optimizer to make it better at deciding when to do a full join and when using keys.
- You can now use `mysqladmin proc` to display information about your own threads. Only users with the **Process_priv** privilege can get information about all threads.
- Added handling of formats `YYMMDD`, `YYYYMMDD`, `YYMMDDHHMMSS` for numbers when using `DATETIME` and `TIMESTAMP` types. (Formerly these formats only worked with strings.)
- Added connect option `CLIENT_IGNORE_SPACE` to allow use of spaces after function names and before '`(`' (Powerbuilder requires this). This will make all function names reserved words.
- Added the `--log-long-format` option to `mysqld` to enable timestamps and INSERT_ID's in the update log.
- Added `--where` option to `mysqldump` (patch by Jim Faucette).
- The lexical analyzer now uses "perfect hashing" for faster parsing of SQL statements.

D.2.21 Changes in release 3.22.6

- Faster `mysqldump`.

- For the `LOAD DATA INFILE` statement, you can now use the new `LOCAL` keyword to read the file from the client. `mysqlimport` will automatically use `LOCAL` when importing with the TCP/IP protocol.
- Fixed small optimize problem when updating keys.
- Changed makefiles to support shared libraries.
- **MySQL**-NT can now use named pipes, which means that you can now use **MySQL**-NT without having to install TCP/IP.

D.2.22 Changes in release 3.22.5

- All table lock handing is changed to avoid some very subtle deadlocks when using `DROP TABLE`, `ALTER TABLE`, `DELETE FROM TABLE` and `mysqladmin flush-tables` under heavy usage. Changed locking code to get better handling of locks of different types.
- Updated `DBI` to 1.00 and `DBD` to 1.2.0.
- Added a check that the error message file contains error messages suitable for the current version of `mysqld`. (To avoid errors if you accidentally try to use an old error message file.)
- All count structures in the client (`affected_rows()`, `insert_id()`,...) are now of type `BIGINT` to allow 64-bit values to be used. This required a minor change in the **MySQL** protocol which should affect only old clients when using tables with `AUTO_INCREMENT` values > 24M.
- The return type of `mysql_fetch_lengths()` has changed from `uint *` to `ulong *`. This may give a warning for old clients but should work on most machines.
- Change `mysys` and `dbug` libraries to allocate all thread variables in one struct. This makes it easier to make a threaded '`libmysql.dll`' library.
- Use the result from `gethostname()` (instead of `uname()`) when constructing '`.pid`' file names.
- New better compressed server/client protocol.
- `COUNT()`, `STD()` and `AVG()` are extended to handle more than 4G rows.
- You can now store values in the range `-838:59:59` <= x <= `838:59:59` in a `TIME` column.
- **WARNING: INCOMPATIBLE CHANGE!!** If you set a `TIME` column to too short a value, **MySQL** now assumes the value is given as: `[[[D ]HH:]MM:]SS` instead of `HH[:MM[:SS]]`.
- `TIME_TO_SEC()` and `SEC_TO_TIME()` can now handle negative times and hours up to 32767.
- Added new option `SET OPTION SQL_LOG_UPDATE={0|1}` to allow users with the **process** privilege to bypass the update log. (Modified patch from Sergey A Mukhin violet@rosnet.net.)
- Fixed fatal bug in `LPAD()`.
- Initialize line buffer in '`mysql.cc`' to make `BLOB` reading from pipes safer.
- Added `-O max_connect_errors=#` option to `mysqld`. Connect errors are now reset for each correct connection.

- Increased the default value of `max_allowed_packet` to `1M` in `mysqld`.
- Added `--low-priority-updates` option to `mysqld`, to give `UPDATE` operations lower priority than retrievals. You can now use `{INSERT | REPLACE | UPDATE | DELETE} LOW_PRIORITY ...` You can also use `SET OPTION LOW_PRIORITY_UPDATES={0|1}` to change the priority for one thread. One side effect is that `LOW_PRIORITY` is now a reserved word. :(
- Add support for `INSERT INTO table ... VALUES(...),(...),(...)`, to allow inserting multiple rows with a single statement.
- `INSERT INTO tbl_name` is now also cached when used with `LOCK TABLES`. (Previously only `INSERT ... SELECT` and `LOAD DATA INFILE` were cached.)
- Allow `GROUP BY` functions with `HAVING`:

```
mysql> SELECT col FROM table GROUP BY col HAVING COUNT(*)>0;
```

- `mysqld` will now ignore trailing '`;`' characters in queries. This is to make it easier to migrate from some other SQL servers that require the trailing '`;`'.
- Fix for corrupted fixed-format output generated by `SELECT INTO OUTFILE`.
- **WARNING: INCOMPATIBLE CHANGE!!** Added Oracle `GREATEST()` and `LEAST()` functions. You must now use these instead of the `MAX()` and `MIN()` functions to get the largest/smallest value from a list of values. These can now handle `REAL`, `BIGINT` and string (`CHAR` or `VARCHAR`) values.
- **WARNING: INCOMPATIBLE CHANGE!!** `DAYOFWEEK()` had offset 0 for Sunday. Changed the offset to 1.
- Give an error for queries that mix `GROUP BY` columns and fields when there is no `GROUP BY` specification.
- Added `--vertical` option to `mysql`, for printing results in vertical mode.
- Index-only optimization; some queries are now resolved using only indexes. Until **MySQL** 4.0, this works only for numeric columns. See Section 10.4 [**MySQL** indexes], page 249.
- Lots of new benchmarks.
- A new C API chapter and lots of other improvements in the manual.

D.2.23 Changes in release 3.22.4

- Added `--tmpdir` option to `mysqld`, for specifying the location of the temporary file directory.
- **MySQL** now automatically changes a query from an ODBC client:

```
SELECT ... FROM table WHERE auto_increment_column IS NULL
```

  to:

```
SELECT ... FROM table WHERE auto_increment_column == LAST_INSERT_ID()
```

  This allows some ODBC programs (Delphi, Access) to retrieve the newly inserted row to fetch the `AUTO_INCREMENT` id.
- `DROP TABLE` now waits for all users to free a table before deleting it.
- Fixed small memory leak in the new connect protocol.

- New functions `BIN()`, `HEX()` and `CONV()` for converting between different number bases.
- Added function `SUBSTRING()` with 2 arguments.
- If you created a table with a record length smaller than 5, you couldn't delete rows from the table.
- Added optimization to remove const reference tables from `ORDER BY` and `GROUP BY`.
- `mysqld` now automatically disables system locking on Linux and Win32, and for systems that use MIT-pthreads. You can force the use of locking with the `--enable-locking` option.
- Added `--console` option to `mysqld`, to force a console window (for error messages) when using Win32.
- Fixed table locks for Win32.
- Allow '`$`' in identifiers.
- Changed name of user-specific configuration file from '`my.cnf`' to '`.my.cnf`' (Unix only).
- Added `DATE_ADD()` and `DATE_SUB()` functions.

D.2.24 Changes in release 3.22.3

- Fixed a lock problem (bug in **MySQL** 3.22.1) when closing temporary tables.
- Added missing `mysql_ping()` to the client library.
- Added `--compress` option to all **MySQL** clients.
- Changed `byte` to `char` in '`mysql.h`' and '`mysql_com.h`'.

D.2.25 Changes in release 3.22.2

- Searching on multiple constant keys that matched more than 30% of the rows didn't always use the best possible key.
- New functions `<<`, `>>`, `RPAD()` and `LPAD()`.
- You can now save default options (like passwords) in a configuration file ('`my.cnf`').
- Lots of small changes to get `ORDER BY` to work when no records are found when using fields that are not in `GROUP BY` (**MySQL** extension).
- Added `--chroot` option to `mysqld`, to start `mysqld` in a chroot environment (by Nikki Chumakov nikkic@cityline.ru).
- Trailing spaces are now ignored when comparing case-sensitive strings; this should fix some problems with ODBC and flag 512!
- Fixed a core-dump bug in the range optimizer.
- Added `--one-thread` option to `mysqld`, for debugging with LinuxThreads (or `glibc`). (This replaces the `-T32` flag)
- Added `DROP TABLE IF EXISTS` to prevent an error from occurring if the table doesn't exist.
- `IF` and `EXISTS` are now reserved words (they would have to be sooner or later).
- Added lots of new options to `mysqldump`.

- Server error messages are now in '`mysqld_error.h`'.
- The server/client protocol now supports compression.
- All bug fixes from **MySQL** 3.21.32.

D.2.26 Changes in release 3.22.1

- Added new C API function `mysql_ping()`.
- Added new API functions `mysql_init()` and `mysql_options()`. You now MUST call `mysql_init()` before you call `mysql_real_connect()`. You don't have to call `mysql_init()` if you only use `mysql_connect()`.
- Added `mysql_options(...,MYSQL_OPT_CONNECT_TIMEOUT,...)` so you can set a timeout for connecting to a server.
- Added `--timeout` option to `mysqladmin`, as a test of `mysql_options()`.
- Added `AFTER column` and `FIRST` options to `ALTER TABLE ... ADD columns`. This makes it possible to add a new column at some specific location within a row in an existing table.
- `WEEK()` now takes an optional argument to allow handling of weeks when the week starts on Monday (some European countries). By default, `WEEK()` assumes the week starts on Sunday.
- `TIME` columns weren't stored properly (bug in **MySQL** 3.22.0).
- `UPDATE` now returns information about how many rows were matched and updated, and how many "warnings" occurred when doing the update.
- Fixed incorrect result from `FORMAT(-100,2)`.
- `ENUM` and `SET` columns were compared in binary (case-sensitive) fashion; changed to be case insensitive.

D.2.27 Changes in release 3.22.0

- New (backward compatible) connect protocol that allows you to specify the database to use when connecting, to get much faster connections to a specific database.

  The `mysql_real_connect()` call is changed to:

  ```
  mysql_real_connect(MYSQL *mysql, const char *host, const char *user,
                     const char *passwd, const char *db, uint port,
                     const char *unix_socket, uint client_flag)
  ```

- Each connection is handled by its own thread, rather than by the master `accept()` thread. This fixes permanently the telnet bug that was a topic on the mail list some time ago.
- All TCP/IP connections are now checked with backward resolution of the hostname to get better security. `mysqld` now has a local hostname resolver cache so connections should actually be faster than before, even with this feature.
- A site automatically will be blocked from future connections if someone repeatedly connects with an "improper header" (like when one uses telnet).

- You can now refer to tables in different databases with references of the form `tbl_name@db_name` or `db_name.tbl_name`. This makes it possible to give a user read access to some tables and write access to others simply by keeping them in different databases!
- Added `--user` option to `mysqld`, to allow it to run as another Unix user (if it is started as the Unix `root` user).
- Added caching of users and access rights (for faster access rights checking)
- Normal users (not anonymous ones) can change their password with `mysqladmin password 'new_password'`. This uses encrypted passwords that are not logged in the normal **MySQL** log!
- All important string functions are now coded in assembler for x86 Linux machines. This gives a speedup of 10% in many cases.
- For tables that have many columns, the column names are now hashed for much faster column name lookup (this will speed up some benchmark tests a lot!)
- Some benchmarks are changed to get better individual timing. (Some loops were so short that a specific test took < 2 seconds. The loops have been changed to take about 20 seconds to make it easier to compare different databases. A test that took 1-2 seconds before now takes 11-24 seconds, which is much better)
- Re-arranged `SELECT` code to handle some very specific queries involving group functions (like `COUNT(*)`) without a `GROUP BY` but with `HAVING`. The following now works:

```
mysql> SELECT count(*) as C FROM table HAVING C > 1;
```

- Changed the protocol for field functions to be faster and avoid some calls to `malloc()`.
- Added `-T32` option to `mysqld`, for running all queries under the main thread. This makes it possible to debug `mysqld` under Linux with `gdb`!
- Added optimization of `not_null_column IS NULL` (needed for some Access queries).
- Allow `STRAIGHT_JOIN` to be used between two tables to force the optimizer to join them in a specific order.
- String functions now return `VARCHAR` rather than `CHAR` and the column type is now `VARCHAR` for fields saved as `VARCHAR`. This should make the **MyODBC** driver better, but may break some old **MySQL** clients that don't handle `FIELD_TYPE_VARCHAR` the same way as `FIELD_TYPE_CHAR`.
- `CREATE INDEX` and `DROP INDEX` are now implemented through `ALTER TABLE`. `CREATE TABLE` is still the recommended (fast) way to create indexes.
- Added `--set-variable` option `wait_timeout` to `mysqld`.
- Added time column to `mysqladmin processlist` to show how long a query has taken or how long a thread has slept.
- Added lots of new variables to `show variables` and some new to `show status`.
- Added new type `YEAR`. `YEAR` is stored in 1 byte with allowable values of 0, and 1901 to 2155.
- Added new `DATE` type that is stored in 3 bytes rather than 4 bytes. All new tables are created with the new date type if you don't use the `--old-protocol` option to `mysqld`.
- Fixed bug in record caches; for some queries, you could get `Error from table handler: #` on some operating systems.

- Added `--enable-assembler` option to `configure`, for x86 machines (tested on Linux + `gcc`). This will enable assembler functions for the most important string functions for more speed!

D.3 Changes in release 3.21.x

D.3.1 Changes in release 3.21.33

- Fixed problem when sending `SIGHUP` to `mysqld`; `mysqld` core dumped when starting from boot on some systems.
- Fixed problem with losing a little memory for some connections.
- `DELETE FROM tbl_name` without a `WHERE` condition is now done the long way when you use `LOCK TABLES` or if the table is in use, to avoid race conditions.
- `INSERT INTO TABLE (timestamp_column) VALUES (NULL);` didn't set timestamp.

D.3.2 Changes in release 3.21.32

- Fixed some possible race conditions when doing many reopen/close on the same tables under heavy load! This can happen if you execute `mysqladmin refresh` often. This could in some very rare cases corrupt the header of the index file and cause error 126 or 138.
- Fixed fatal bug in `refresh()` when running with the `--skip-locking` option. There was a "very small" time gap after a `mysqladmin refresh` when a table could be corrupted if one thread updated a table while another thread did `mysqladmin refresh` and another thread started a new update ont the same table before the first thread had finished. A refresh (or `--flush-tables`) will now not return until all used tables are closed!
- `SELECT DISTINCT` with a `WHERE` clause that didn't match any rows returned a row in some contexts (bug only in 3.21.31).
- `GROUP BY + ORDER BY` returned one empty row when no rows where found.
- Fixed a bug in the range optimizer that wrote `Use_count: Wrong count for ...` in the error log file.

D.3.3 Changes in release 3.21.31

- Fixed a sign extension problem for the `TINYINT` type on Irix.
- Fixed problem with `LEFT("constant_string",function)`.
- Fixed problem with `FIND_IN_SET()`.
- `LEFT JOIN` core dumped if the second table is used with a constant `WHERE/ON` expression that uniquely identifies one record.
- Fixed problems with `DATE_FORMAT()` and incorrect dates. `DATE_FORMAT()` now ignores `'%'` to make it possible to extend it more easily in the future.

D.3.4 Changes in release 3.21.30

- `mysql` now returns an exit code > 0 if the query returned an error.
- Saving of command line history to file in `mysql` client. By Tommy Larsen tommy@mix.hive.no.
- Fixed problem with empty lines that were ignored in '`mysql.cc`'.
- Save the pid of the signal handler thread in the pid file instead of the pid of the main thread.
- Added patch by tommy@valley.ne.jp to support Japanese characters SJIS and UJIS.
- Changed `safe_mysqld` to redirect startup messages to `'hostname'.err` instead of `'hostname'.log` to reclaim file space on `mysqladmin refresh`.
- `ENUM` always had the first entry as default value.
- `ALTER TABLE` wrote two entries to the update log.
- `sql_acc()` now closes the `mysql` grant tables after a reload to save table space and memory.
- Changed `LOAD DATA` to use less memory with tables and `BLOB` columns.
- Sorting on a function which made a division / 0 produced a wrong set in some cases.
- Fixed `SELECT` problem with `LEFT()` when using the czech character set.
- Fixed problem in `isamchk`; it couldn't repair a packed table in a very unusual case.
- `SELECT` statements with `&` or `|` (bit functions) failed on columns with `NULL` values.
- When comparing a field = field, where one of the fields was a part key, only the length of the part key was compared.

D.3.5 Changes in release 3.21.29

- `LOCK TABLES + DELETE from tbl_name` never removed locks properly.
- Fixed problem when grouping on an `OR` function.
- Fixed permission problem with `umask()` and creating new databases.
- Fixed permission problem on result file with `SELECT ... INTO OUTFILE ...`
- Fixed problem in range optimizer (core dump) for a very complex query.
- Fixed problem when using `MIN(integer)` or `MAX(integer)` in `GROUP BY`.
- Fixed bug on Alpha when using integer keys. (Other keys worked on Alpha).
- Fixed bug in `WEEK("XXXX-xx-01")`.

D.3.6 Changes in release 3.21.28

- Fixed socket permission (clients couldn't connect to Unix socket on Linux).
- Fixed bug in record caches; for some queries, you could get `Error from table handler: #` on some operating systems.

D.3.7 Changes in release 3.21.27

- Added user level lock functions `GET_LOCK(string,timeout)`, `RELEASE_LOCK(string)`.
- Added `opened_tables` to `show status`.
- Changed connect timeout to 3 seconds to make it somewhat harder for crackers to kill `mysqld` through telnet + TCP/IP.
- Fixed bug in range optimizer when using `WHERE key_part_1 >= something AND key_part_2 <= something_else`.
- Changed `configure` for detection of FreeBSD 3.0 9803xx and above
- `WHERE` with string_column_key = constant_string didn't always find all rows if the column had many values differing only with characters of the same sort value (like e and é).
- Strings keys looked up with 'ref' were not compared in case-sensitive fashion.
- Added `umask()` to make log files non-readable for normal users.
- Ignore users with old (8-byte) password on startup if not using `--old-protocol` option to `mysqld`.
- `SELECT` which matched all key fields returned the values in the case of the matched values, not of the found values. (Minor problem.)

D.3.8 Changes in release 3.21.26

- `FROM_DAYS(0)` now returns "0000-00-00".
- In `DATE_FORMAT()`, PM and AM were swapped for hours 00 and 12.
- Extended the default maximum key size to 256.
- Fixed bug when using `BLOB/TEXT` in `GROUP BY` with many tables.
- An `ENUM` field that is not declared `NOT NULL` has `NULL` as the default value. (Previously, the default value was the first enumeration value.)
- Fixed bug in the join optimizer code when using many part keys on the same key: `INDEX (Organization,Surname(35),Initials(35))`.
- Added some tests to the table order optimizer to get some cases with `SELECT ... FROM many_tables` much faster.
- Added a retry loop around `accept()` to possibly fix some problems on some Linux machines.

D.3.9 Changes in release 3.21.25

- Changed `typedef 'string'` to `typedef 'my_string'` for better portability.
- You can now kill threads that are waiting on a disk full condition.
- Fixed some problems with UDF functions.
- Added long options to `isamchk`. Try `isamchk --help`.
- Fixed a bug when using 8 bytes long (alpha); `filesort()` didn't work. Affects `DISTINCT`, `ORDER BY` and `GROUP BY` on 64-bit processors.

D.3.10 Changes in release 3.21.24

- Dynamic loadable functions. Based on source from Alexis Mikhailov.
- You couldn't delete from a table if no one had done a `SELECT` on the table.
- Fixed problem with range optimizer with many `OR` operators on key parts inside each other.
- Recoded `MIN()` and `MAX()` to work properly with strings and `HAVING`.
- Changed default umask value for new files from `0664` to `0660`.
- Fixed problem with `LEFT JOIN` and constant expressions in the `ON` part.
- Added Italian error messages from `brenno@3cord.philips.nl`.
- `configure` now works better on OSF1 (tested on 4.0D).
- Added hooks to allow `LIKE` optimization with international character support.
- Upgraded `DBI` to 0.93.

D.3.11 Changes in release 3.21.23

- The following symbols are now reserved words: `TIME`, `DATE`, `TIMESTAMP`, `TEXT`, `BIT`, `ENUM`, `NO`, `ACTION`, `CHECK`, `YEAR`, `MONTH`, `DAY`, `HOUR`, `MINUTE`, `SECOND`, `STATUS`, `VARIABLES`.
- Setting a `TIMESTAMP` to `NULL` in `LOAD DATA INFILE ...` didn't set the current time for the `TIMESTAMP`.
- Fix `BETWEEN` to recognize binary strings. Now `BETWEEN` is case sensitive.
- Added `--skip-thread-priority` option to `mysqld`, for systems where `mysqld`'s thread scheduling doesn't work properly (BSDI 3.1).
- Added ODBC functions `DAYNAME()` and `MONTHNAME()`.
- Added function `TIME_FORMAT()`. This works like `DATE_FORMAT()`, but takes a time string (`'HH:MM:DD'`) as argument.
- Fixed unlikely(?) key optimizer bug when using `OR`s of key parts inside `AND`s.
- Added command `variables` to `mysqladmin`.
- A lot of small changes to the binary releases.
- Fixed a bug in the new protocol from **MySQL** 3.21.20.
- Changed `ALTER TABLE` to work with Win32 (Win32 can't rename open files). Also fixed a couple of small bugs in the Win32 version.
- All standard **MySQL** clients are now ported to MySQL-Win32.
- **MySQL** can now be started as a service on NT.

D.3.12 Changes in release 3.21.22

- Starting with this version, all **MySQL** distributions will be configured, compiled and tested with `crash-me` and the benchmarks on the following platforms: SunOS 5.6 sun4u, SunOS 5.5.1 sun4u, SunOS 4.14 sun4c, SunOS 5.6 i86pc, Irix 6.3 mips5k, HP-UX 10.20 hppa, AIX 4.2.1 ppc, OSF1 V4.0 alpha, FreeBSD 2.2.2 i86pc and BSDI 3.1 i386.

- Fix `COUNT(*)` problems when the `WHERE` clause didn't match any records. (Bug from 3.21.17.)
- Removed that `NULL = NULL` is true. Now you must use `IS NULL` or `IS NOT NULL` to test whether or not a value is `NULL`. (This is according to ANSI SQL but may break old applications that are ported from `mSQL`.) You can get the old behavior by compiling with `-DmSQL_COMPLIANT`.
- Fixed bug that core dumped when using many `LEFT OUTER JOIN` clauses.
- Fixed bug in `ORDER BY` on string formula with possible `NULL` values.
- Fixed problem in range optimizer when using `<=` on sub index.
- Added functions `DAYOFYEAR()`, `DAYOFMONTH()`, `MONTH()`, `YEAR()`, `WEEK()`, `QUARTER()`, `HOUR()`, `MINUTE()`, `SECOND()` and `FIND_IN_SET()`.
- Added command `SHOW VARIABLES`.
- Added support of "long constant strings" from ANSI SQL:

```
mysql> SELECT 'first ' 'second';       -> 'first second'
```

- Upgraded mSQL-Mysql-modules to 1.1825.
- Upgraded `mysqlaccess` to 2.02.
- Fixed problem with Russian character set and `LIKE`.
- Ported to OpenBSD 2.1.
- New Dutch error messages.

D.3.13 Changes in release 3.21.21a

- Configure changes for some operating systems.

D.3.14 Changes in release 3.21.21

- Fixed optimizer bug when using `WHERE data_field = date_field2 AND date_field2 = constant`.
- Added command `SHOW STATUS`.
- Removed '`manual.ps`' from the source distribution to make it smaller.

D.3.15 Changes in release 3.21.20

- Changed the maximum table name and column name lengths from 32 to 64.
- Aliases can now be of "any" length.
- Fixed `mysqladmin stat` to return the right number of queries.
- Changed protocol (downward compatible) to mark if a column has the `AUTO_INCREMENT` attribute or is a `TIMESTAMP`. This is needed for the new Java driver.
- Added Hebrew sorting order by Zeev Suraski.
- Solaris 2.6: Fixed `configure` bugs and increased maximum table size from 2G to 4G.

D.3.16 Changes in release 3.21.19

- Upgraded DBD to 1823. This version implements mysql_use_result in DBD-Mysql.
- Benchmarks updated for empress (by Luuk).
- Fixed a case of slow range searching.
- Configure fixes ('Docs' directory).
- Added function REVERSE() (by Zeev Suraski).

D.3.17 Changes in release 3.21.18

- Issue error message if client C functions are called in wrong order.
- Added automatic reconnect to the 'libmysql.c' library. If a write command fails, an automatic reconnect is done.
- Small sort sets no longer use temporary files.
- Upgraded DBI to 0.91.
- Fixed a couple of problems with LEFT OUTER JOIN.
- Added CROSS JOIN syntax. CROSS is now a reserved word.
- Recoded yacc/bison stack allocation to be even safer and to allow **MySQL** to handle even bigger expressions.
- Fixed a couple of problems with the update log.
- ORDER BY was slow when used with key ranges.

D.3.18 Changes in release 3.21.17

- Changed documentation string of --with-unix-socket-path to avoid confusion.
- Added ODBC and ANSI SQL style LEFT OUTER JOIN.
- The following are new reserved words: LEFT, NATURAL, USING.
- The client library now uses the value of the environment variable MYSQL_HOST as the default host if it's defined.
- SELECT column, SUM(expr) now returns NULL for column when there are matching rows.
- Fixed problem with comparing binary strings and BLOBs with ASCII characters over 127.
- Fixed lock problem: when freeing a read lock on a table with multiple read locks, a thread waiting for a write lock would have been given the lock. This shouldn't affect data integrity, but could possibly make mysqld restart if one thread was reading data that another thread modified.
- LIMIT offset,count didn't work in INSERT ... SELECT.
- Optimized key block caching. This will be quicker than the old algorithm when using bigger key caches.

D.3.19 Changes in release 3.21.16

- Added ODBC 2.0 & 3.0 functions `POWER()`, `SPACE()`, `COT()`, `DEGREES()`, `RADIANS()`, `ROUND(2 arg)` and `TRUNCATE()`.
- **WARNING: INCOMPATIBLE CHANGE!!** `LOCATE()` parameters were swapped according to ODBC standard. Fixed.
- Added function `TIME_TO_SEC()`.
- In some cases, default values were not used for `NOT NULL` fields.
- Timestamp wasn't always updated properly in `UPDATE SET ...` statements.
- Allow empty strings as default values for `BLOB` and `TEXT`, to be compatible with `mysqldump`.

D.3.20 Changes in release 3.21.15

- **WARNING: INCOMPATIBLE CHANGE!!** `mysqlperl` is now from Msql-Mysql-modules. This means that `connect()` now takes `host`, `database`, `user`, `password` arguments! The old version took `host`, `database`, `password`, `user`.
- Allow `DATE '1997-01-01'`, `TIME '12:10:10'` and `TIMESTAMP '1997-01-01 12:10:10'` formats required by ANSI SQL. **WARNING: INCOMPATIBLE CHANGE!!** This has the unfortunate side-effect that you no longer can have columns named `DATE`, `TIME` or `TIMESTAMP`. :(Old columns can still be accessed through `tablename.columnname`!)
- Changed Makefiles to hopefully work better with BSD systems. Also, '`manual.dvi`' is now included in the distribution to avoid having stupid `make` programs trying to rebuild it.
- `readline` library upgraded to version 2.1.
- A new sortorder german-1. That is a normal ISO-Latin1 with a german sort order.
- Perl `DBI/DBD` is now included in the distribution. `DBI` is now the recommended way to connect to **MySQL** from Perl.
- New portable benchmark suite with `DBD`, with test results from `mSQL` 2.0.3, **MySQL**, PostgreSQL 6.2.1 and Solid server 2.2.
- `crash-me` is now included with the benchmarks; This is a Perl program designed to find as many limits as possible in a SQL server. Tested with `mSQL`, PostgreSQL, Solid and **MySQL**.
- Fixed bug in range-optimizer that crashed **MySQL** on some queries.
- Table and column name completion for `mysql` command line tool, by Zeev Suraski and Andi Gutmans.
- Added new command `REPLACE` that works like `INSERT` but replaces conflicting records with the new record. `REPLACE INTO TABLE ... SELECT ...` works also.
- Added new commands `CREATE DATABASE db_name` and `DROP DATABASE db_name`.
- Added `RENAME` option to `ALTER TABLE`: `ALTER TABLE name RENAME AS new_name`.
- `make_binary_distribution` now includes '`libgcc.a`' in '`libmysqlclient.a`'. This should make linking work for people who don't have `gcc`.

- Changed `net_write()` to `my_net_write()` because of a name conflict with Sybase.
- New function `DAYOFWEEK()` compatible with ODBC.
- Stack checking and `bison` memory overrun checking to make **MySQL** safer with weird queries.

D.3.21 Changes in release 3.21.14b

- Fixed a couple of small `configure` problems on some platforms.

D.3.22 Changes in release 3.21.14a

- Ported to SCO Openserver 5.0.4 with FSU Pthreads.
- HP-UX 10.20 should work.
- Added new function `DATE_FORMAT()`.
- Added `NOT IN`.
- Added automatic removal of 'ODBC function conversions': `{fn now() }`
- Handle ODBC 2.50.3 option flags.
- Fixed comparison of `DATE` and `TIME` values with `NULL`.
- Changed language name from germany to german to be consistent with the other language names.
- Fixed sorting problem on functions returning a `FLOAT`. Previously, the values were converted to `INT`s before sorting.
- Fixed slow sorting when sorting on key field when using `key_column=constant`.
- Sorting on calculated `DOUBLE` values sorted on integer results instead.
- `mysql` no longer needs a database argument.
- Changed the place where `HAVING` should be. According to ANSI, it should be after `GROUP BY` but before `ORDER BY`. **MySQL** 3.20 incorrectly had it last.
- Added Sybase command `USE DATABASE` to start using another database.
- Added automatic adjusting of number of connections and table cache size if the maximum number of files that can be opened is less than needed. This should fix that `mysqld` doesn't crash even if you haven't done a `ulimit -n 256` before starting `mysqld`.
- Added lots of limit checks to make it safer when running with too little memory or when doing weird queries.

D.3.23 Changes in release 3.21.13

- Added retry of interrupted reads and clearing of `errno`. This makes Linux systems much safer!
- Fixed locking bug when using many aliases on the same table in the same `SELECT`.
- Fixed bug with `LIKE` on number key.
- New error message so you can check whether the connection was lost while the command was running or whether the connection was down from the start.

- Added `--table` option to `mysql` to print in table format. Moved time and row information after query result. Added automatic reconnect of lost connections.
- Added `!=` as a synonym for `<>`.
- Added function `VERSION()` to make easier logs.
- New multi-user test '`tests/fork_test.pl`' to put some strain on the thread library.

D.3.24 Changes in release 3.21.12

- Fixed `ftruncate()` call in MIT-pthreads. This made `isamchk` destroy the '`.ISM`' files on (Free)BSD 2.x systems.
- Fixed broken `__P_` patch in MIT-pthreads.
- Many memory overrun checks. All string functions now return `NULL` if the returned string should be longer than `max_allowed_packet` bytes.
- Changed the name of the `INTERVAL` type to `ENUM`, because `INTERVAL` is used in ANSI SQL.
- In some cases, doing a `JOIN + GROUP + INTO OUTFILE`, the result wasn't grouped.
- `LIKE` with `'_'` as last character didn't work. Fixed.
- Added extended ANSI SQL `TRIM()` function.
- Added `CURTIME()`.
- Added `ENCRYPT()` function by Zeev Suraski.
- Fixed better `FOREIGN KEY` syntax skipping. New reserved words: `MATCH`, `FULL`, `PARTIAL`.
- `mysqld` now allows IP number and hostname to the `--bind-address` option.
- Added `SET OPTION CHARACTER SET cp1251_koi8` to enable conversions of data to/from cp1251_koi8.
- Lots of changes for Win95 port. In theory, this version should now be easily portable to Win95.
- Changed the `CREATE COLUMN` syntax of `NOT NULL` columns to be after the `DEFAULT` value, as specified in the ANSI SQL standard. This will make `mysqldump` with `NOT NULL` and default values incompatible with **MySQL** 3.20.
- Added many function name aliases so the functions can be used with ODBC or ANSI SQL92 syntax.
- Fixed syntax of `ALTER TABLE tbl_name ALTER COLUMN col_name SET DEFAULT NULL`.
- Added `CHAR` and `BIT` as synonyms for `CHAR(1)`.
- Fixed core dump when updating as a user who has only **select** privilege.
- `INSERT ... SELECT ... GROUP BY` didn't work in some cases. An `Invalid use of group function` error occurred.
- When using `LIMIT`, `SELECT` now always uses keys instead of record scan. This will give better performance on `SELECT` and a `WHERE` that matches many rows.
- Added Russian error messages.

D.3.25 Changes in release 3.21.11

- Configure changes.
- **MySQL** now works with the new thread library on BSD/OS 3.0.
- Added new group functions `BIT_OR()` and `BIT_AND()`.
- Added compatibility functions `CHECK` and `REFERENCES`. `CHECK` is now a reserved word.
- Added `ALL` option to `GRANT` for better compatibility. (`GRANT` is still a dummy function.)
- Added partly-translated dutch messages.
- Fixed bug in `ORDER BY` and `GROUP BY` with `NULL` columns.
- Added function `last_insert_id()` to retrieve last `AUTO_INCREMENT` value. This is intended for clients to ODBC that can't use the `mysql_insert_id()` API function, but can be used by any client.
- Added `--flush-logs` option to `mysqladmin`.
- Added command `STATUS` to `mysql`.
- Fixed problem with `ORDER BY/GROUP BY` because of bug in `gcc`.
- Fixed problem with `INSERT ... SELECT ... GROUP BY`.

D.3.26 Changes in release 3.21.10

- New `mysqlaccess`.
- `CREATE` now supports all ODBC types and the `mSQL` `TEXT` type. All ODBC 2.5 functions are also supported (added `REPEAT`). This provides better portability.
- Added text types `TINYTEXT`, `TEXT`, `MEDIUMTEXT` and `LONGTEXT`. These are actually `BLOB`types, but all searching is done in case-insensitive fashion.
- All old `BLOB` fields are now `TEXT` fields. This only changes that all searching on strings is done in case-sensitive fashion. You must do an `ALTER TABLE` and change the field type to `BLOB` if you want to have tests done in case-sensitive fashion.
- Fixed some `configure` issues.
- Made the locking code a bit safer. Fixed very unlikely deadlock situation.
- Fixed a couple of bugs in the range optimizer. Now the new range benchmark `test-select` works.

D.3.27 Changes in release 3.21.9

- Added `--enable-unix-socket=pathname` option to `configure`.
- Fixed a couple of portability problems with include files.
- Fixed bug in range calculation that could return empty set when searching on multiple key with only one entry (very rare).
- Most things ported to FSU Pthreads, which should allow **MySQL** to run on SCO. See Section 4.11.12 [SCO], page 61.

D.3.28 Changes in release 3.21.8

- Works now in Solaris 2.6.
- Added handling of calculation of `SUM()` functions. For example, you can now use `SUM(column)/COUNT(column)`.
- Added handling of trigometric functions: `PI()`, `ACOS()`, `ASIN()`, `ATAN()`, `COS()`, `SIN()` and `TAN()`.
- New languages: norwegian, norwegian-ny and portuguese.
- Fixed parameter bug in `net_print()` in '`procedure.cc`'.
- Fixed a couple of memory leaks.
- Now allow also the old `SELECT ... INTO OUTFILE` syntax.
- Fixed bug with `GROUP BY` and `SELECT` on key with many values.
- `mysql_fetch_lengths()` sometimes returned incorrect lengths when you used `mysql_use_result()`. This affected at least some cases of `mysqldump --quick`.
- Fixed bug in optimization of `WHERE const op field`.
- Fixed problem when sorting on `NULL` fields.
- Fixed a couple of 64-bit (Alpha) problems.
- Added `--pid-file=#` option to `mysqld`.
- Added date formatting to `FROM_UNIXTIME()`, originally by Zeev Suraski.
- Fixed bug in `BETWEEN` in range optimizer (Did only test = of the first argument).
- Added machine-dependent files for MIT-pthreads i386-SCO. There is probably more to do to get this to work on SCO 3.5.

D.3.29 Changes in release 3.21.7

- Changed '`Makefile.am`' to take advantage of Automake 1.2.
- Added the beginnings of a benchmark suite.
- Added more secure password handling.
- Added new client function `mysql_errno()`, to get the error number of the error message. This makes error checking in the client much easier. This makes the new server incompatible with the 3.20.x server when running without `--old-protocol`. The client code is backward compatible. More information can be found in the '`README`' file!
- Fixed some problems when using very long, illegal names.

D.3.30 Changes in release 3.21.6

- Fixed more portability issues (incorrect `sigwait` and `sigset` defines).
- `configure` should now be able to detect the last argument to `accept()`.

D.3.31 Changes in release 3.21.5

- Should now work with FreeBSD 3.0 if used with '`FreeBSD-3.0-libc_r-1.0.diff`', which can be found at http://www.mysql.com/Download/Patches.
- Added new option `-O tmp_table_size=#` to `mysqld`.
- New function `FROM_UNIXTIME(timestamp)` which returns a date string in 'YYYY-MM-DD HH:MM:DD' format.
- New function `SEC_TO_TIME(seconds)` which returns a string in 'HH:MM:SS' format.
- New function `SUBSTRING_INDEX()`, originally by Zeev Suraski.

D.3.32 Changes in release 3.21.4

- Should now configure and compile on OSF1 4.0 with the DEC compiler.
- Configuration and compilation on BSD/OS 3.0 works, but due to some bugs in BSD/OS 3.0, `mysqld` doesn't work on it yet.
- Configuration and compilation on FreeBSD 3.0 works, but I couldn't get `pthread_create` to work.

D.3.33 Changes in release 3.21.3

- Added reverse check lookup of hostnames to get better security.
- Fixed some possible buffer overflows if filenames that are too long are used.
- `mysqld` doesn't accept hostnames that start with digits followed by a `'.'`, because the hostname may look like an IP number.
- Added `--skip-networking` option to `mysqld`, to only allow socket connections. (This will not work with MIT-pthreads!)
- Added check of too long table names for alias.
- Added check if database name is okay.
- Added check if too long table names.
- Removed incorrect `free()` that killed the server on `CREATE DATABASE` or `DROP DATABASE`.
- Changed some `mysqld -O` options to better names.
- Added `-O join_cache_size=#` option to `mysqld`.
- Added `-O max_join_size=#` option to `mysqld`, to be able to set a limit how big queries (in this case big = slow) one should be able to handle without specifying `SET OPTION SQL_BIG_SELECTS=1`. A # = is about 10 examined records. The default is "unlimited".
- When comparing a `TIME`, `DATE`, `DATETIME` or `TIMESTAMP` column to a constant, the constant is converted to a time value before performing the comparison. This will make it easier to get ODBC (particularly Access97) to work with the above types. It should also make dates easier to use and the comparisons should be quicker than before.
- Applied patch from Jochen Wiedmann that allows `query()` in `mysqlperl` to take a query with `\0` in it.

- Storing a timestamp with a 2-digit year (`YYMMDD`) didn't work.
- Fix that timestamp wasn't automatically updated if set in an `UPDATE` clause.
- Now the automatic timestamp field is the FIRST timestamp field.
- `SELECT * INTO OUTFILE`, which didn't correctly if the outfile already existed.
- `mysql` now shows the thread ID when starting or doing a reconnect.
- Changed the default sort buffer size from 2M to 1M.

D.3.34 Changes in release 3.21.2

- The range optimizer is coded, but only 85% tested. It can be enabled with `--new`, but it crashes core a lot yet...
- More portable. Should compile on AIX and alpha-digital. At least the `isam` library should be relatively 64-bit clean.
- New `isamchk` which can detect and fix more problems.
- New options for `isamlog`.
- Using new version of Automake.
- Many small portability changes (from the AIX and alpha-digital port) Better checking of pthread(s) library.
- czech error messages by snajdr@pvt.net.
- Decreased size of some buffers to get fewer problems on systems with little memory. Also added more checks to handle "out of memory" problems.
- `mysqladmin`: you can now do `mysqladmin kill 5,6,7,8` to kill multiple threads.
- When the maximum connection limit is reached, one extra connection by a user with the **PROCESS_ACL** privilege is granted.
- Added `-O backlog=#` option to `mysqld`.
- Increased maximum packet size from 512K to 1024K for client.
- Almost all of the function code is now tested in the internal test suite.
- `ALTER TABLE` now returns warnings from field conversions.
- Port changed to 3306 (got it reserved from ISI).
- Added a fix for Visual FoxBase so that any schema name from a table specification is automatically removed.
- New function `ASCII()`.
- Removed function `BETWEEN(a,b,c)`. Use the standard ANSI synax instead: `expr BETWEEN expr AND expr`.
- **MySQL** no longer has to use an extra temporary table when sorting on functions or `SUM()` functions.
- Fixed bug that you couldn't use `tbl_name.field_name` in `UPDATE`.
- Fixed `SELECT DISTINCT` when using 'hidden group'. For example:

```
mysql> SELECT DISTINCT MOD(some_field,10) FROM test
           GROUP BY some_field;
```

  Note: `some_field` is normally in the `SELECT` part. ANSI SQL should require it.

D.3.35 Changes in release 3.21.0

- New keywords used: `INTERVAL`, `EXPLAIN`, `READ`, `WRITE`, `BINARY`.
- Added ODBC function `CHAR(num,...)`.
- New operator `IN`. This uses a binary search to find a match.
- New command `LOCK TABLES tbl_name [AS alias] {READ|WRITE} ...`
- Added `--log-update` option to `mysqld`, to get a log suitable for incremental updates.
- New command `EXPLAIN SELECT ...` to get information about how the optimizer will do the join.
- For easier client code, the client should no longer use `FIELD_TYPE_TINY_BLOB`, `FIELD_TYPE_MEDIUM_BLOB`, `FIELD_TYPE_LONG_BLOB` or `FIELD_TYPE_VAR_STRING` (as previously returned by `mysql_list_fields`). You should instead only use `FIELD_TYPE_BLOB` or `FIELD_TYPE_STRING`. If you want exact types, you should use the command `SHOW FIELDS`.
- Added varbinary syntax: `0x######` which can be used as a string (default) or a number.
- `FIELD_TYPE_CHAR` is renamed to `FIELD_TYPE_TINY`.
- Changed all fields to C++ classes.
- Removed FORM struct.
- Fields with `DEFAULT` values no longer need to be `NOT NULL`.
- New field types:

  `ENUM` A string which can take only a couple of defined values. The value is stored as a 1-3 byte number that is mapped automatically to a string. This is sorted according to string positions!

  `SET` A string which may have one or many string values separated with ','. The string is stored as a 1-, 2-, 3-, 4- or 8-byte number where each bit stands for a specific set member. This is sorted according to the unsigned value of the stored packed number.
- Now all function calculation is done with `double` or `long long`. This will provide the full 64-bit range with bit functions and fix some conversions that previously could result in precision losses. One should avoid using `unsigned long long` columns with full 64-bit range (numbers bigger than 9223372036854775807) because calculations are done with `signed long long`.
- `ORDER BY` will now put `NULL` field values first. `GROUP BY` will also work with `NULL` values.
- Full `WHERE` with expressions.
- New range optimizer that can resolve ranges when some keypart prefix is constant. Example:

```
mysql> SELECT * FROM tbl_name
           WHERE key_part_1="customer"
           AND key_part_2>=10 AND key_part_2<=10;
```

D.4 Changes in release 3.20.x

Changes from 3.20.18 to 3.20.32b are not documented here since the 3.21 release branched here. And the relevant changes are also documented as changes to the 3.21 version.

D.4.1 Changes in release 3.20.18

- Added `-p#` (remove `#` directories from path) to `isamlog`. All files are written with a relative path from the database directory Now `mysqld` shouldn't crash on shutdown when using the `--log-isam` option.
- New `mysqlperl` version. It is now compatible with `msqlperl-0.63`.
- New `DBD` module available at http://www.mysql.com/Contrib site.
- Added group function `STD()` (standard deviation).
- The `mysqld` server is now compiled by default without debugging information. This will make the daemon smaller and faster.
- Now one usually only has to specify the `--basedir` option to `mysqld`. All other paths are relative in a normal installation.
- `BLOB` columns sometimes contained garbage when used with a `SELECT` on more than one table and `ORDER BY`.
- Fixed that calculations that are not in `GROUP BY` work as expected (ANSI SQL extension). Example:

```
mysql> SELECT id,id+1 FROM table GROUP BY id;
```

- The test of using `MYSQL_PWD` was reversed. Now `MYSQL_PWD` is enabled as default in the default release.
- Fixed conversion bug which caused `mysqld` to core dump with Arithmetic error on Sparc-386.
- Added `--unbuffered` option to `mysql`, for new `mysqlaccess`.
- When using overlapping (unnecessary) keys and join over many tables, the optimizer could get confused and return 0 records.

D.4.2 Changes in release 3.20.17

- You can now use `BLOB` columns and the functions `IS NULL` and `IS NOT NULL` in the `WHERE` clause.
- All communication packets and row buffers are now allocated dynamically on demand. The default value of `max_allowed_packet` is now 64K for the server and 512K for the client. This is mainly used to catch incorrect packets that could trash all memory. The server limit may be changed when it is started.
- Changed stack usage to use less memory.
- Changed `safe_mysqld` to check for running daemon.
- The `ELT()` function is renamed to `FIELD()`. The new `ELT()` function returns a value based on an index: `FIELD()` is the inverse of `ELT()` Example: `ELT(2,"A","B","C")` returns `"B"`. `FIELD("B","A","B","C")` returns 2.

- `COUNT(field)`, where `field` could have a `NULL` value, now works.
- A couple of bugs fixed in `SELECT ... GROUP BY`.
- Fixed memory overrun bug in `WHERE` with many unoptimizable brace levels.
- Fixed some small bugs in the grant code.
- If hostname isn't found by `get_hostname`, only the IP is checked. Previously, you got `Access denied`.
- Inserts of timestamps with values didn't always work.
- `INSERT INTO ... SELECT ... WHERE` could give the error `Duplicated field`.
- Added some tests to `safe_mysqld` to make it "safer".
- `LIKE` was case sensitive in some places and case insensitive in others. Now `LIKE` is always case insensitive.
- '`mysql.cc`': Allow `'#'` anywhere on the line.
- New command `SET OPTION SQL_SELECT_LIMIT=#`. See the FAQ for more details.
- New version of the `mysqlaccess` script.
- Change `FROM_DAYS()` and `WEEKDAY()` to also take a full `TIMESTAMP` or `DATETIME` as argument. Before they only took a number of type `YYYYMMDD` or `YYMMDD`.
- Added new function `UNIX_TIMESTAMP(timestamp_column)`.

D.4.3 Changes in release 3.20.16

- More changes in MIT-pthreads to get them safer. Fixed also some link bugs at least in SunOS.
- Changed `mysqld` to work around a bug in MIT-pthreads. This makes multiple small `SELECT` operations 20 times faster. Now `lock_test.pl` should work.
- Added `mysql_FetchHash(handle)` to `mysqlperl`.
- The `mysqlbug` script is now distributed built to allow for reporting bugs that appear during the build with it.
- Changed '`libmysql.c`' to prefer `getpwuid()` instead of `cuserid()`.
- Fixed bug in `SELECT` optimizer when using many tables with the same column used as key to different tables.
- Added new latin2 and Russian KOI8 character tables.
- Added support for a dummy `GRANT` command to satisfy Powerbuilder.

D.4.4 Changes in release 3.20.15

- Fixed fatal bug `packets out of order` when using MIT-pthreads.
- Removed possible loop when a thread waits for command from client and `fcntl()` fails. Thanks to Mike Bretz for finding this bug.
- Changed alarm loop in '`mysqld.cc`' because shutdown didn't always succeed in Linux.
- Removed use of `termbits` from '`mysql.cc`'. This conflicted with `glibc` 2.0.
- Fixed some syntax errors for at least BSD and Linux.

- Fixed bug when doing a SELECT as superuser without a database.
- Fixed bug when doing SELECT with group calculation to outfile.

D.4.5 Changes in release 3.20.14

- If one gives -p or --password option to mysql without an argument, the user is solicited for the password from the tty.
- Added default password from MYSQL_PWD (by Elmar Haneke).
- Added command kill to mysqladmin to kill a specific **MySQL** thread.
- Sometimes when doing a reconnect on a down connection this succeeded first on second try.
- Fixed adding an AUTO_INCREMENT key with ALTER_TABLE.
- AVG() gave too small value on some SELECTs with GROUP BY and ORDER BY.
- Added new DATETIME type (by Giovanni Maruzzelli maruzz@matrice.it).
- Fixed that define DONT_USE_DEFAULT_FIELDS works.
- Changed to use a thread to handle alarms instead of signals on Solaris to avoid race conditions.
- Fixed default length of signed numbers. (George Harvey georgeh@pinacl.co.uk.)
- Allow anything for CREATE INDEX.
- Add prezeros when packing numbers to DATE, TIME and TIMESTAMP.
- Fixed a bug in OR of multiple tables (gave empty set).
- Added many patches to MIT-pthreads. This fixes at least one lookup bug.

D.4.6 Changes in release 3.20.13

- Added ANSI SQL94 DATE and TIME types.
- Fixed bug in SELECT with AND-OR levels.
- Added support for Slovenian characters. The 'Contrib' directory contains source and instructions for adding other character sets.
- Fixed bug with LIMIT and ORDER BY.
- Allow ORDER BY and GROUP BY on items that aren't in the SELECT list. (Thanks to Wim Bonis bonis@kiss.de, for pointing this out.)
- Allow setting of timestamp values in INSERT.
- Fixed bug with SELECT ... WHERE ... = NULL.
- Added changes for glibc 2.0. To get glibc to work, you should add the 'gibc-2.0-sigwait-patch' before compiling glibc.
- Fixed bug in ALTER TABLE when changing a NOT NULL field to allow NULL values.
- Added some ANSI92 synonyms as field types to CREATE TABLE. CREATE TABLE now allows FLOAT(4) and FLOAT(8) to mean FLOAT and DOUBLE.
- New utility program mysqlaccess by Yves.Carlier@rug.ac.be. This program shows the access rights for a specific user and the grant rows that determine this grant.
- Added WHERE const op field (by bonis@kiss.de).

D.4.7 Changes in release 3.20.11

- When using `SELECT ... INTO OUTFILE`, all temporary tables are ISAM instead of HEAP to allow big dumps.
- Changed date functions to be string functions. This fixed some "funny" side effects when sorting on dates.
- Extended `ALTER TABLE` according to SQL92.
- Some minor compability changes.
- Added `--port` and `--socket` options to all utility programs and `mysqld`.
- Fixed MIT-pthreads `readdir_r()`. Now `mysqladmin create database` and `mysqladmin drop database` should work.
- Changed MIT-pthreads to use our `tempnam()`. This should fix the "sort aborted" bug.
- Added sync of records count in `sql_update`. This fixed slow updates on first connection. (Thanks to Vaclav Bittner for the test.)

D.4.8 Changes in release 3.20.10

- New insert type: `INSERT INTO ... SELECT ...`
- `MEDIUMBLOB` fixed.
- Fixed bug in `ALTER TABLE` and `BLOB`s.
- `SELECT ... INTO OUTFILE` now creates the file in the current database directory.
- `DROP TABLE` now can take a list of tables.
- Oracle synonym `DESCRIBE` (`DESC`).
- Changes to `make_binary_distribution`.
- Added some comments to installation instructions about `configure`'s C++ link test.
- Added `--without-perl` option to `configure`.
- Lots of small portability changes.

D.4.9 Changes in release 3.20.9

- `ALTER TABLE` didn't copy null bit. As a result, fields that were allowed to have `NULL` values were always `NULL`.
- `CREATE` didn't take numbers as `DEFAULT`.
- Some compatibility changes for SunOS.
- Removed '`config.cache`' from old distribution.

D.4.10 Changes in release 3.20.8

- Fixed bug with `ALTER TABLE` and multi-part keys.

D.4.11 Changes in release 3.20.7

- New commands: `ALTER TABLE`, `SELECT ... INTO OUTFILE` and `LOAD DATA INFILE`.
- New function: `NOW()`.
- Added new field **file_priv** to `mysql/user` table.
- New script `add_file_priv` which adds the new field **file_priv** to the `user` table. This script must be executed if you want to use the new `SELECT ... INTO` and `LOAD DATA INFILE ...` commands with a version of **MySQL** earlier than 3.20.7.
- Fixed bug in locking code, which made `lock_test.pl` test fail.
- New files '`NEW`' and '`BUGS`'.
- Changed '`select_test.c`' and '`insert_test.c`' to include '`config.h`'.
- Added command `status` to `mysqladmin` for short logging.
- Increased maximum number of keys to 16 and maximum number of key parts to 15.
- Use of sub keys. A key may now be a prefix of a string field.
- Added `-k` option to `mysqlshow`, to get key information for a table.
- Added long options to `mysqldump`.

D.4.12 Changes in release 3.20.6

- Portable to more systems because of MIT-pthreads, which will be used automatically if `configure` cannot find a `-lpthreads` library.
- Added GNU-style long options to almost all programs. Test with *program* `--help`.
- Some shared library support for Linux.
- The FAQ is now in '`.texi`' format and is available in '`.html`', '`.txt`' and '`.ps`' formats.
- Added new SQL function `RAND([init])`.
- Changed `sql_lex` to handle `\0` unquoted, but the client can't send the query through the C API, because it takes a str pointer. You must use `mysql_real_query()` to send the query.
- Added API function `mysql_get_client_info()`.
- `mysqld` now uses the `N_MAX_KEY_LENGTH` from '`nisam.h`' as the maximum allowed key length.
- The following now works:

  ```
  mysql> SELECT filter_nr,filter_nr FROM filter ORDER BY filter_nr;
  ```

  Previously, this resulted in the error: `Column: 'filter_nr' in order clause is ambiguous`.
- `mysql` now outputs `'\0'`, `'\t'`, `'\n'` and `'\\'` when encountering ASCII 0, tab, newline or `'\'` while writing tab-separated output. This is to allow printing of binary data in a portable format. To get the old behavior, use `-r` (or `--raw`).
- Added german error messages (60 of 80 error messages translated).
- Added new API function `mysql_fetch_lengths(MYSQL_RES *)`, which returns an array of of column lengths (of type `uint`).

- Fixed bug with `IS NULL` in `WHERE` clause.
- Changed the optimizer a little to get better results when searching on a key part.
- Added `SELECT` option `STRAIGHT_JOIN` to tell the optimizer that it should join tables in the given order.
- Added support for comments starting with `'--'` in '`mysql.cc`' (Postgres syntax).
- You can have `SELECT` expressions and table columns in a `SELECT` which are not used in the group part. This makes it efficient to implement lookups. The column that is used should be a constant for each group because the value is calculated only once for the first row that is found for a group.

```
mysql> SELECT id,lookup.text,sum(*) FROM test,lookup
          WHERE test.id=lookup.id GROUP BY id;
```

- Fixed bug in `SUM(function)` (could cause a core dump).
- Changed `AUTO_INCREMENT` placement in the SQL query:

```
INSERT into table (auto_field) values (0);
```

  inserted 0, but it should insert an `AUTO_INCREMENT` value.
- '`mysqlshow.c`': Added number of records in table. Had to change the client code a little to fix this.
- `mysql` now allows doubled `''` or `""` within strings for embedded `'` or `"`.
- New math functions: `EXP()`, `LOG()`, `SQRT()`, `ROUND()`, `CEILING()`.

D.4.13 Changes in release 3.20.3

- The `configure` source now compiles a thread-free client library `-lmysqlclient`. This is the only library that needs to be linked with client applications. When using the binary releases, you must link with `-lmysql -lmysys -ldbug -lstrings` as before.
- New `readline` library from `bash-2.0`.
- LOTS of small changes to `configure` and makefiles (and related source).
- It should now be possible to compile in another directory using `VPATH`. Tested with GNU Make 3.75.
- `safe_mysqld` and `mysql.server` changed to be more compatible between the source and the binary releases.
- `LIMIT` now takes one or two numeric arguments. If one argument is given, it indicates the maximum number of rows in a result. If two arguments are given, the first argument indicates the offset of the first row to return, the second is the maximum number of rows. With this it's easy to do a poor man's next page/previous page WWW application.
- Changed name of SQL function `FIELDS()` to `ELT()`. Changed SQL function `INTERVALL()` to `INTERVAL()`.
- Made `SHOW COLUMNS` a synonym for `SHOW FIELDS`. Added compatibility syntax `FRIEND KEY` to `CREATE TABLE`. In **MySQL**, this creates a non-unique key on the given columns.
- Added `CREATE INDEX` and `DROP INDEX` as compatibility functions. In **MySQL**, `CREATE INDEX` only checks if the index exists and issues an error if it doesn't exist. `DROP INDEX` always succeeds.

- 'mysqladmin.c': added client version to version information.
- Fixed core dump bug in sql_acl (core on new connection).
- Removed host, user and db tables from database test in the distribution.
- FIELD_TYPE_CHAR can now be signed (-128 - 127) or unsigned (0 - 255) Previously, it was always unsigned.
- Bug fixes in CONCAT() and WEEKDAY().
- Changed a lot of source to get mysqld to be compiled with SunPro compiler.
- SQL functions must now have a '(' immediately after the function name (no intervening space). For example, 'user(' is regarded as beginning a function call, and 'user (' is regarded as an identifier user followed by a '(', not as a function call.

D.4.14 Changes in release 3.20.0

- The source distribution is done with configure and Automake. It will make porting much easier. The readline library is included in the distribution.
- Separate client compilation: the client code should be very easy to compile on systems which don't have threads.
- The old Perl interface code is automatically compiled and installed. Automatic compiling of DBD will follow when the new DBD code is ported.
- Dynamic language support: mysqld can now be started with Swedish or English (default) error messages.
- New functions: INSERT(), RTRIM(), LTRIM() and FORMAT().
- mysqldump now works correctly for all field types (even AUTO_INCREMENT). The format for SHOW FIELDS FROM tbl_name is changed so the Type column contains information suitable for CREATE TABLE. In previous releases, some CREATE TABLE information had to be patched when recreating tables.
- Some parser bugs from 3.19.5 (BLOB and TIMESTAMP) are corrected. TIMESTAMP now returns different date information depending on its create length.
- Changed parser to allow a database, table or field name to start with a number or '_'.
- All old C code from Unireg changed to C++ and cleaned up. This makes the daemon a little smaller and easier to understand.
- A lot of small bug fixes done.
- New INSTALL files (not final version) and some info regarding porting.

D.5 Changes in release 3.19.x

D.5.1 Changes in release 3.19.5

- Some new functions, some more optimization on joins.
- Should now compile clean on Linux (2.0.x).
- Added functions DATABASE(), USER(), POW(), LOG10() (needed for ODBC).

- In a `WHERE` with an `ORDER BY` on fields from only one table, the table is now preferred as first table in a multi-join.
- `HAVING` and `IS NULL` or `IS NOT NULL` now works.
- A group on one column and a sort on a group function (`SUM()`, `AVG()`...) didn't work together. Fixed.
- `mysqldump`: Didn't send password to server.

D.5.2 Changes in release 3.19.4

- Fixed horrible locking bug when inserting in one thread and reading in another thread.
- Fixed one-off decimal bug. 1.00 was output as 1.0.
- Added attribute `'Locked'` to process list as info if a query is locked by another query.
- Fixed full magic timestamp. Timestamp length may now be 14, 12, 10, 8, 6, 4 or 2 bytes.
- Sort on some numeric functions could sort incorrectly on last number.
- `IF(arg,syntax_error,syntax_error)` crashed.
- Added functions `CEILING()`, `ROUND()`, `EXP()`, `LOG()` and `SQRT()`.
- Enhanced `BETWEEN` to handle strings.

D.5.3 Changes in release 3.19.3

- Fixed `SELECT` with grouping on `BLOB` columns not to return incorrect `BLOB` info. Grouping, sorting and distinct on `BLOB` columns will not yet work as expected (probably it will group/sort by the first 7 characters in the `BLOB`). Grouping on formulas with a fixed string size (use `MID()` on a `BLOB`) should work.
- When doing a full join (no direct keys) on multiple tables with `BLOB` fields, the `BLOB` was garbage on output.
- Fixed `DISTINCT` with calculated columns.

Appendix E Known errors and design deficiencies in MySQL

- You cannot build in another directory when using MIT-pthreads. Since this requires changes to MIT-pthreads, we are not likely to fix this.
- `BLOB` values can't "reliably" be used in `GROUP BY` or `ORDER BY` or `DISTINCT`. Only the first `max_sort_length` bytes (default 1024) are used when comparing `BLOB`bs in these cases. This can be changed with the `-O max_sort_length` option to `mysqld`. A workaround for most cases is to use a substring: `SELECT DISTINCT LEFT(blob,2048) FROM tbl_name`.
- Calculation is done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). It depends on the function which precision one gets. The general rule is that bit functions are done with `BIGINT` precision, `IF`, and `ELT()` with `BIGINT` or `DOUBLE` precision and the rest with `DOUBLE` precision. One should try to avoid using bigger unsigned long long values than 63 bits (9223372036854775807) for anything else than bit fields!
- Before **MySQL** 3.23 all numeric types where treated as fixed-point fields. That means you had to specify how many decimals a floating-point field shall have. All results was returned with the correct number of decimals.
- All string columns, except `BLOB` and `TEXT` columns, automatically have all trailing spaces removed when retrieved. For `CHAR` types this is okay, and may be regarded as a feature according to ANSI SQL92. The bug is that in **MySQL**, `VARCHAR` columns are treated the same way.
- You can only have up to 255 `ENUM` and `SET` columns in one table.
- Before **MySQL** 3.23.2 an `UPDATE` that updated a key with a `WHERE` on the same key may have failed because the key was used to search for records and the same row may have been found multiple times:

  ```
  UPDATE tbl_name SET KEY=KEY+1 WHERE KEY > 100;
  ```

  A workaround is to use:

  ```
  mysql> UPDATE tbl_name SET KEY=KEY+1 WHERE KEY+0 > 100;
  ```

  This will work because **MySQL** will not use index on expressions in the `WHERE` clause.
- `safe_mysqld` re-directs all messages from `mysqld` to the `mysqld` log. One problem with this is that if you execute `mysqladmin refresh` to close and reopen the log, `stdout` and `stderr` are still redirected to the old log. If you use `--log` extensively, you should edit `safe_mysqld` to log to '`'hostname'.err`' instead of '`'hostname'.log`' so you can easily reclaim the space for the old log by deleting the old one and executing `mysqladmin refresh`.
- In the `UPDATE` statement, columns are updated from left to right. If you refer to a updated column, you will get the updated value instead of the original value. For example:

  ```
  mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1
  ```

  will update `KEY` with `2` instead of with `1`.

For platform-specific bugs, see the sections about compiling and porting.

Appendix F List of things we want to add to MySQL in the future (The TODO)

Everything in this list is in the order it will be done. If you want to affect the priority order, please register a license or support us and tell us what you want to have done more quickly. See Chapter 3 [Licensing and Support], page 20.

F.1 Things that must done in the real near future

Subqueries. `select id from t where grp in (select grp from g where u > 100)`

- If you perform an `ALTER TABLE` on a table that is symlinked to another disk, create temporary tables on this disk.
- `RENAME table as table, table as table [,...]`
- FreeBSD and MIT-pthreads; Do sleeping threads take CPU?
- Allow join on key parts (optimization issue).
- Entry for `DECRYPT()`.
- Remember `FOREIGN` key definitions in the '`.frm`' file.
- Server side cursors.
- Allow users to change startup options.
- Add –ansi option to MySQL (to change `||` -> CONCAT())
- Don't add automatic `DEFAULT` values to columns. Give an error when using an `INSERT` that doesn't contain a column that doesn't have a `DEFAULT`.
- Caching of queries and results. This should be done as a separated module that examines each query and if this is query is in the cache the cached result should be returned. When one updates a table one should remove as few queries as possible from the cache. This should give a big speed bost on machines with much RAM where queries are often repeated (like WWW applications). One idea would be to only cache queries of type: `SELECT CACHED ....`
- Fix '`libmysql.c`' to allow two `mysql_query()` commands in a row without reading results or give a nice error message when one does this.
- Optimize `BIT` type to take 1 bit (now `BIT` takes 1 char).
- Check why MIT-pthreads `ctime()` doesn't work on some FreeBSD systems.
- Check if locked threads take any CPU.
- Add `ORDER BY` to update. This would be handy with functions like: `generate_id(start,step)`.
- Add an `IMAGE` option to `LOAD DATA INFILE` to not update `TIMESTAMP` and `AUTO_INCREMENT` fields.
- Make `LOAD DATA INFILE` understand a syntax like:

```
LOAD DATA INFILE 'file_name.txt' INTO TABLE tbl_name
TEXT_FIELDS (text_field1, text_field2, text_field3)
SET table_field1=concatenate(text_field1, text_field2), table_field3=23
IGNORE text_field3
```

- Allow strings with `MIN()`, `MAX()` (not group functions). These should be synonyms for `LEAST()`, `GREATEST()`.
- Demo procedure: analyze
- Automatic output from `mysql` to netscape.
- `LOCK DATABASES`. (with various options)
- `NATURAL JOIN`.
- Change sort to allocate memory in "hunks" to get better memory utilization.
- `DECIMAL` and `NUMERIC` types can't read exponential numbers; `Field_decimal::store(const char *from,uint len)` must be recoded to fix this.
- Fix `mysql.cc` to do fewer `malloc()` calls when hashing field names.
- Functions: ADD_TO_SET(value,set) and REMOVE_FROM_SET(value,set)
- Add use of `t1 JOIN t2 ON ...` and `t1 JOIN t2 USING ...` Currently, you can only use this syntax with `LEFT JOIN`.
- Add full support for `unsigned long long` type.
- Function `CASE`.
- Many more variables for `show status`. Counts for: `INSERT/DELETE/UPDATE` statements. Records reads and updated. Selects on 1 table and selects with joins. Mean number of tables in select. Key buffer read/write hits (logical and real). `ORDER BY`, `GROUP BY`, temporary tables created.
- If you abort `mysql` in the middle of a query, you should open another connection and kill the old running query. Alternatively, an attempt should be made to detect this in the server.
- Add a handler interface for table information so you can use it as a system table. This would be a bit slow if you requested information about all tables, but very flexible. `SHOW INFO FROM tbl_name` for basic table information should be implemented.
- Allow `mysqld` to support many character sets at the same time.
- Add support for UNICODE.
- `NATURAL JOIN`.
- Oracle like `CONNECT BY PRIOR ...` to search hierarchy structures.
- `RENAME DATABASE`
- `mysqladmin copy database new-database`.
- Processlist should show number of queries/thread.
- `IGNORE` option to the `UPDATE` statement (this will delete all rows that gets a dupplicate key error while updating).
- `select distinct a from foo order by b` should not return duplicate rows, if there are different pairs of (a,b)
- Change the format of `DATETIME` to store fractions of seconds.
- Add all missing ANSI92 and ODBC 3.0 types.
- Change table names from empty strings to `NULL` for calculated columns.

F.2 Things that have to be done sometime

- Implement function: `get_changed_tables(timeout,table1,table2,...)`
- Implement function: `LAST_UPDATED(tbl_name)`
- Atomic updates; This includes a language that one can even use for a set of stored procedures.
- `update items,month set items.price=month.price where items.id=month.id;`
- Change reading through tables to use memmap when possible. Now only compressed tables use memmap.
- Add a new privilege **'Show_priv'** for `SHOW` commands.
- Make the automatic timestamp code nicer. Add timestamps to the update log with `SET TIMESTAMP=#;`
- Use read/write mutex in some places to get more speed.
- Full foreign key support. One probably wants to implement a procedural language first.
- Simple views (first on one table, later on any expression).
- Automatically close some tables if a table, temporary table or temporary files gets error 23 (not enough open files).
- When one finds a field=#, change all occurrences of field to #. Now this is only done for some simple cases.
- Change all const expressions with calculated expressions if possible.
- Optimize key = expression. At the moment only key = field or key = constant are optimized.
- Join some of the copy functions for nicer code.
- Change '`sql_yacc.yy`' to an inline parser to reduce its size and get better error messages (5 days).
- Change the parser to use only one rule per different number of arguments in function.
- Use of full calculation names in the order part. (For ACCESS97)
- `UNION`, `MINUS`, `INTERSECT` and `FULL OUTER JOIN`. (Currently only `LEFT OUTER JOIN` is supported)
- Allow `UNIQUE` on fields that can be `NULL`.
- `SQL_OPTION MAX_SELECT_TIME=#` to put a time limit on a query.
- Make the update log to a database.
- Negative `LIMIT` to retrieve data from the end.
- Alarm around client connect/read/write functions.
- Make a `mysqld` version which isn't multithreaded (3-5 days).
- Please note the changes to `safe_mysqld`: according to FSSTND (which Debian tries to follow) PID files should go into '`/var/run/<progname>.pid`' and log files into '`/var/log`'. It would be nice if you could put the "DATADIR" in the first declaration of "pidfile" and "log", so the placement of these files can be changed with a single statement.

- Better dynamic record layout to avoid fragmentation.
- `UPDATE SET blob=read_blob_from_file('my_gif') where id=1;`
- Allow a client to request logging.
- Add use of `zlib()` for `gzip`-ed files to `LOAD DATA INFILE`.
- Fix sorting and grouping of `BLOB` columns (partly solved now).
- Stored procedures. This is currently not regarded to be very important as stored procedures are not very standardized yet. Another problem is that true stored procedures make it much harder for the optimizer and in many cases the result is slower than before We will, on the other hand, add a simple (atomic) update language that can be used to write loops and such in the **MySQL** server.
- Change to use semaphores when counting threads. One should first implement a semaphore library to MIT-pthreads.
- Don't assign a new `AUTO_INCREMENT` value when one sets a column to 0. Use `NULL` instead.
- Add full support for `JOIN` with parentheses.
- Reuse threads for system with a lot of connections.

Time is given according to amount of work, not real time. TcX's main business is the use of **MySQL** not the development of it. But since TcX is a very flexible company, we have put a lot of resources into the development of **MySQL**.

F.3 Some things we don't have any plans to do

- Transactions with rollback (we mainly do `SELECT`s, and because we don't do transactions, we can be much quicker on everything else). We will support some kind of atomic operations on multiple tables, though. Currently atomic operations can be done with `LOCK TABLES`/`UNLOCK TABLES` but we will make this more automatic in the future.

Appendix G Comments on porting to other systems

A working Posix thread library is needed for the server. On Solaris 2.5 we use SUN PThreads (the native thread support in 2.4 and earlier versions are not good enough) and on Linux we use LinuxThreads by Xavier Leroy, Xavier.Leroy@inria.fr.

The hard part of porting to a new Unix variant without good native thread support is probably to port MIT-pthreads. See '`mit-pthreads/README`' and Programming POSIX Threads.

The **MySQL** distribution includes a patched version of Provenzano's Pthreads from MIT (see MIT Pthreads web page). This can be used for some operating systems that do not have POSIX threads.

It is also possible to use another user level thread package named FSU Pthreads (see FSU Pthreads home page). This implementation is being used for the SCO port.

See the '`thr_lock.c`' and '`thr_alarm.c`' programs in the '`mysys`' directory for some tests/examples of these problems.

Both the server and the client need a working C++ compiler (we use `gcc` and have tried SparcWorks). Another compiler that is known to work is the Irix `cc`.

To compile only the client use `./configure --without-server`.

There is currently no support for only compiling the server. Nor is it likly to be added unless someone has a good reason for it.

If you want/need to change any '`Makefile`' or the configure script you must get Automake and Autoconf. We have used the `automake-1.2` and `autoconf-2.12` distributions.

All steps needed to remake everything from the most basic files.

```
/bin/rm */.deps/*.P
/bin/rm -f config.cache
aclocal
autoheader
aclocal
automake
autoconf
./configure --with-debug --prefix='your installation directory'

# The makefiles generated above need GNU make 3.75 or newer.
# (called gmake below)
gmake clean all install init-db
```

If you run into problems with a new port, you may have to do some debugging of **MySQL**! See Section G.1 [Debugging server], page 444.

Note: Before you start debugging `mysqld`, first get the test programs `mysys/thr_alarm` and `mysys/thr_lock` to work. This will ensure that your thread installation has even a remote chance to work!

G.1 Debugging a MySQL server

If you are using some functionality that is very new in **MySQL**, you can try to run mysqld with the `--skip-new` (which will disable all new, potentially unsafe functionality) or with `--safe-mode` which disables a lot of optimization that may cause problems. See Section 18.1 [Crashing], page 307.

If `mysqld` doesn't want to start, you should check that you don't have any `my.cnf` file that interferes with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`.

If you have some very specific problem, you can always try to debug **MySQL**. To do this you must configure **MySQL** with the option `--with-debug`. You can check whether or not **MySQL** was compiled with debugging by doing: `mysqld --help`. If the `--debug` flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql ... -debug` in this case.

If you are using gcc or egcs, the recommended configure line is:

```
CC=gcc CFLAGS="-O6" CXX=gcc CXXFLAGS="-O6 -felide-constructors -fno-exceptions -fno
```

This will avoid problems with the libstdc++ library and with C++ exceptions.

If you can cause the `mysqld` server to crash quickly, you can try to create a trace file of this:

Start the `mysqld` server with a trace log in '`/tmp/mysql.trace`'. The log file will get very *BIG*.

```
mysqld --debug --log
```

or you can start it with

```
mysqld --debug=d,info,error,query,general,where:O,/tmp/mysql.trace
```

which only prints information with the most interesting tags.

When you configure **MySQL** for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something "unexpected," an entry will be written to `stderr`, which `safe_mysqld` directs to the error log! This also means that if you are having some unexpected problems with **MySQL** and are using a source distribution, the first thing you should do is to configure **MySQL** for debugging! (The second thing, of course, is to send mail to `mysql@lists.mysql.com` and ask for help. Please use the `mysqlbug` script for all bug reports or questions regarding the **MySQL** version you are using!

On most system you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case you can only have one thread active at a time.

If you are using gdb 4.17.x on Linux, you should install a '`.gdb`' file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
```

```
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

Here follows an example how to debug mysqld:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
back   # Do this when mysqld crashes
info locals
up
info locals
up
...
(until you get some information about local variables)

quit
```

Include the above output in a mail generated with `mysqlbug` and mail this to `mysql@lists.mysql.com`.

If `mysqld` hangs you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hanged.

If `mysqld` starts to eat up CPU or memory or if it "hangs", you can use `mysqladmin processlist status` to find out if someone is executing some query that takes a long time. It may be a good idea to run `mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients can't connect.

If `mysqld` dies or hangs, you should start `mysqld` with `--log`. When `mysqld` dies again, you can check in the log file for the query that killed `mysqld`. Note that before starting `mysqld` with `--log` you should check all your tables with `isamchk`. See Chapter 13 [Maintenance], page 280.

If you are using a log file, `mysqld --log`, you should check the 'hostname' log files, that you can find in the database directory, for any queries that could cause a problem. Try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that mysqld are using indexes properly. See Section 7.21 [EXPLAIN], page 194. You should also test complicated queries that didn't complete within the `mysql` command line tool.

If you find the text `mysqld restarted` in the error log file (normally named '`hostname.err`') you have probably found a query that causes `mysqld` to fail. If this happens you should check all your tables with `isamchk` (see Chapter 13 [Maintenance], page 280), and test the queries in the **MySQL** log files if someone doesn't work. If you find such a query, try first upgrading to the newest **MySQL** version. If this doesn't help and you can't find anything in the `mysql` mail archive, you should report the bug to `mysql@lists.mysql.com`. Links to mail archives are available at the online **MySQL** documentation page.

If you get corrupted tables or if `mysqld` always fails after some update commands, you can test if this bug is reproducible by doing the following:

- Stop the mysqld daemon (with `mysqladmin shutdown`)

- Check all tables with `isamchk -s database/*.ISM`. Repair any wrong tables with `isamchk -r database/table.ISM`.
- Start `mysqld` with `--log-update`
- When you have got a crashed table, stop the `mysqld server`.
- Restore the backup.
- Restart the `mysqld` server **without** `--log-update`
- Re-execute the commands with `mysql < update-log`. The update log is saved in the **MySQL** database directory with the name `your-hostname.#`.
- If the tables are now again corrupted, you have found reproducible bug in the `ISAM` code! ftp the tables + the update log to `ftp://www.mysql.com/pub/mysql/secret` and we will fix this as soon as possible!

The command `mysqladmin debug` will dump some information about locks in use, used memory and query usage to the mysql log file. This may help solve some problems. This command also provides some useful information even if you haven't compiled **MySQL** for debugging!

If the problem is that some tables are getting slower and slower you should try to repair the tables with `isamchk` to optimize the table layout. You should also check the slow queries with `EXPLAIN`. See Chapter 13 [Maintenance], page 280.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See Section 4.11 [Source install system issues], page 51

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable. See Section 20.5.2 [Perl DBI Class], page 368.

G.2 Debugging a MySQL client

To be able to debug a **MySQL** client with the integrated debug package, you should configure **MySQL** with `--with-debug`. See Section 4.7.3 [configure options], page 42

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in '`/tmp/client.trace`'.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming you have compiled **MySQL** with debugging on):

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

This will provide useful information in case you mail a bug report. See Section 2.3 [Bug reports], page 15.

If your client crashes at some 'legal' looking code, you should check that your '`mysql.h`' include file matches your mysql library file. A very common mistake is to use an old '`mysql.h`' file from an old **MySQL** installation with new **MySQL** library.

G.3 Comments about RTS threads

I have tried to use the RTS thread packages with **MySQL** but stumbled on the following problems:

They use old version of a lot of POSIX calls and it is very tedious to make wrappers for all functions. I am inclined to think that it would be easier to change the thread libraries to the newest POSIX specification.

Some wrappers are already written. See '`mysys/my_pthread.c`' for more info.

At least the following should be changed:

`pthread_get_specific` should use one argument. `sigwait` should take two arguments. A lot of functions (at least `pthread_cond_wait`, `pthread_cond_timedwait`) should return the error code on error. Now they return -1 and set `errno`.

Another problem is that user-level threads use the `ALRM` signal and this aborts a lot of functions (`read`, `write`, `open`...). **MySQL** should do a retry on interrupt on all of these but it is not that easy to verify it.

The biggest unsolved problem is the following:

To get thread-level alarms I changed '`mysys/thr_alarm.c`' to wait between alarms with `pthread_cond_timedwait()`, but this aborts with error `EINTR`. I tried to debug the thread library as to why this happens, but couldn't find any easy solution.

If someone wants to try **MySQL** with RTS threads I suggest the following:

- Change functions **MySQL** uses from the thread library to POSIX. This shouldn't take that long.
- Compile all libraries with the `-DHAVE_rts_threads`.
- Compile `thr_alarm`.
- If there are some small differences in the implementation, they may be fixed by changing '`my_pthread.h`' and '`my_pthread.c`'.
- Run `thr_alarm`. If it runs without any "warning", "error" or aborted messages, you are on the right track. Here follows a successful run on Solaris:

```
Main thread: 1
Tread 0 (5) started
Thread: 5  Waiting
process_alarm
Tread 1 (6) started
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 1 (1) sec
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 2 (2) sec
Thread: 6  Simulation of no alarm needed
```

```
Thread: 6  Slept for 0 (3) sec
Thread: 6  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 4 (4) sec
Thread: 6  Waiting
process_alarm
thread_alarm
Thread: 5  Slept for 10 (10) sec
Thread: 5  Waiting
process_alarm
process_alarm
thread_alarm
Thread: 6  Slept for 5 (5) sec
Thread: 6  Waiting
process_alarm
process_alarm

...
thread_alarm
Thread: 5  Slept for 0 (1) sec
end
```

G.4 Differences between different thread packages

MySQL is very dependent on the thread package used. So when choosing a good platform for **MySQL**, the thread package is very important.

There are at least three types of thread packages:

- User threads in a single process. Thread switching is managed with alarms and the threads library manages all non-thread-safe functions with locks. Read, write and select operations are usually managed with a thread-specific select that switches to another thread if the running threads have to wait for data. If the user thread packages are integrated in the standard libs (FreeBSD and BSDI threads) the thread package requires less overhead than thread packages that have to map all unsafe calls (MIT-pthreads, FSU Pthreads and RTS threads). In some environments (for example, SCO), all system calls are thread-safe so the mapping can be done very easily (FSU Pthreads on SCO). Downside: All mapped calls take a little time and it's quite tricky to be able to handle all situations. There are usually also some system calls that are not handled by the thread package (like MIT-pthreads and sockets). Thread scheduling isn't always optimal.
- User threads in separate processes. Thread switching is done by the kernel and all data are shared between threads. The thread package manages the standard thread calls to allow sharing data between threads. LinuxThreads is using this method. Downside: Lots of processes. Thread creating is slow. If one thread dies the rest are usually left

hanging and you must kill them all before restarting. Thread switching is somewhat expensive.

- Kernel threads. Thread switching is handled by the thread library or the kernel and is very fast. Everything is done in one process, but on some systems, `ps` may show the different threads. If one thread aborts the whole process aborts. Most system calls are thread-safe and should require very little overhead. Solaris, HP-UX, AIX and OSF1 have kernel threads.

In some systems kernel threads are managed by integrating user level threads in the system libraries. In such cases, the thread switching can only be done by the thread library and the kernel isn't really "thread aware".

Appendix H Description of MySQL regular expression syntax

A regular expression (regex) is a powerful way of specifying a complex search.

MySQL uses regular Henry Spencer's inplementation of regular expressions. And that is aimed to conform to POSIX 1003.2. **MySQL** uses the extended version.

This is a simplistic reference that skips the details. To get more exact information, see Henry Spencer's `regex(7)` manual page that is included in the source distribution. See Appendix C [Credits], page 394.

A regular expression describes a set of strings. The simplest regexp is one that has no special characters in it. For example, the regexp `hello` matches `hello` and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regexp `hello|word` matches either the string `hello` or the string `word`.

As a more complex example, the regexp `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs` and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression may use any of the following special characters/constructs:

`^` Match the beginning of a string.

```
mysql> select "fo\nfo" REGEXP "^fo$";          -> 0
mysql> select "fofo" REGEXP "^fo";             -> 1
```

`$` Match the end of a string.

```
mysql> select "fo\no" REGEXP "^fo\no$";        -> 1
mysql> select "fo\no" REGEXP "^fo$";           -> 0
```

`.` Match any character (including newline).

```
mysql> select "fofo" REGEXP "^f.*";            -> 1
mysql> select "fo\nfo" REGEXP "^f.*";          -> 1
```

`a*` Match any sequence of zero or more `a` characters.

```
mysql> select "Ban" REGEXP "^Ba*n";            -> 1
mysql> select "Baaan" REGEXP "^Ba*n";          -> 1
mysql> select "Bn" REGEXP "^Ba*n";             -> 1
```

`a+` Match any sequence of one or more `a` characters.

```
mysql> select "Ban" REGEXP "^Ba+n";            -> 1
mysql> select "Bn" REGEXP "^Ba+n";             -> 0
```

`a?` Match either zero or one `a` character.

```
mysql> select "Bn" REGEXP "^Ba?n";             -> 1
mysql> select "Ban" REGEXP "^Ba?n";            -> 1
mysql> select "Baan" REGEXP "^Ba?n";           -> 0
```

`de|abc` Match either of the sequences `de` or `abc`.

```
mysql> select "pi" REGEXP "pi|apa";            -> 1
mysql> select "axe" REGEXP "pi|apa";           -> 0
mysql> select "apa" REGEXP "pi|apa";           -> 1
```

```
mysql> select "apa" REGEXP "^(pi|apa)$";        -> 1
mysql> select "pi" REGEXP "^(pi|apa)$";         -> 1
mysql> select "pix" REGEXP "^(pi|apa)$";        -> 0
```

`(abc)*` Match zero or more instances of the sequence `abc`.

```
mysql> select "pi" REGEXP "^(pi)+$";            -> 1
mysql> select "pip" REGEXP "^(pi)+$";           -> 0
mysql> select "pipi" REGEXP "^(pi)+$";          -> 1
```

`{1}`
`{2,3}` The is a more general way of writing regexps that match many occurrences of the previous atom.

`a*` Can be written as `a{0,}`.

`a+` Can be written as `a{1,}`.

`a?` Can be written as `a{0,1}`.

To be more precise, an atom followed by a bound containing one integer `i` and no comma matches a sequence of exactly `i` matches of the atom. An atom followed by a bound containing one integer `i` and a comma matches a sequence of `i` or more matches of the atom. An atom followed by a bound containing two integers `i` and `j` matches a sequence of `i` through `j` (inclusive) matches of the atom.

Both arguments must `0 >= value <= RE_DUP_MAX (default 255)`. If there are two arguments, the second must be greater than or equal to the first.

`[a-dX]`
`[^a-dX]` Matches any character which is (or is not, if `^` is used) either `a`, `b`, `c`, `d` or `X`. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. So `[0-9]` matches any decimal digit. Any character that does not have a defined meaning inside a `[]` pair has no special meaning and matches only itself.

```
mysql> select "aXbc" REGEXP "[a-dXYZ]";         -> 1
mysql> select "aXbc" REGEXP "^[a-dXYZ]$";       -> 0
mysql> select "aXbc" REGEXP "^[a-dXYZ]+$";      -> 1
mysql> select "aXbc" REGEXP "^[^a-dXYZ]+$";     -> 0
mysql> select "gheis" REGEXP "^[^a-dXYZ]+$";    -> 1
mysql> select "gheisa" REGEXP "^[^a-dXYZ]+$";   -> 0
```

`[[.characters.]]`
The sequence of characters of that collating element. The sequence is a single element of the bracket expression's list. A bracket expression containing a multi-character collating element can thus match more than one character, e.g., if the collating sequence includes a `ch` collating element, then the regular expression `[[.ch.]]*c` matches the first five characters of `chchcc`.

`[=character-class=]`
An equivalence class, standing for the sequences of characters of all collating elements equivalent to that one, including itself.

For example, if `o` and `(+)` are the members of an equivalence class, then `[[=o=]]`, `[[=(+)=]]`, and `[o(+)]` are all synonymous. An equivalence class may not be an endpoint of a range.

`[:character_class:]`

Within a bracket expression, the name of a character class enclosed in `[:` and `:]` stands for the list of all characters belonging to that class. Standard character class names are:

alnum	digit	punct
alpha	graph	space
blank	lower	upper
cntrl	print	xdigit

These stand for the character classes defined in the `ctype(3)` manual page. A locale may provide others. A character class may not be used as an endpoint of a range.

```
mysql> select "justalnums" REGEXP "[[:alnum:]]+";       -> 1
mysql> select "!!" REGEXP "[[:alnum:]]+";               -> 0
```

`[[:<:]]`

`[[:>:]]` These match the null string at the beginning and end of a word respectively. A word is defined as a sequence of word characters which is neither preceded nor followed by word characters. A word character is an alnum character (as defined by `ctype(3)`) or an underscore (`_`).

```
mysql> select "a word a" REGEXP "[[:<:]]word[[:>:]]";      -> 1
mysql> select "a xword a" REGEXP "[[:<:]]word[[:>:]]";     -> 0
```

```
mysql> select "weeknights" REGEXP "^(wee|week)(knights|nights)$"; -> 1
```

Appendix I What is Unireg?

Unireg is our tty interface builder, but it uses a low level connection to our NISAM (which is used by **MySQL**) and because of this it is very quick. It has existed since 1979 (on Unix in C since ~1986).

Unireg has the following components:

- One table viewer with updates/browsing.
- Multi table viewer (with one scrolling region).
- Table creator. (With lots of column tags you can't create with **MySQL**) This is WYSIWYG (for a tty). You design a screen and Unireg prompts for the column specification.
- Report generator.
- A lot of utilities (quick export/import of tables to/from text files, analysis of table contents...).
- Powerful multi-table updates (which we use a lot) with a BASIC-like language with LOTS of functions.
- Dynamic languages (at present in Swedish and Finnish). If somebody wants an English version there are a few files that would have to be translated.
- The ability to run updates interactively or in a batch.
- Emacs-like key definitions with keyboard macros.
- All this in a binary of 800K.
- The `convform` utility. Converts '`.frm`' and text files between different character sets.
- The `pack_isam` utility. Packs a NISAM table (makes it 50-80% smaller). The table can be read by **MySQL** like an ordinary table. Only one record has to be decompressed per access. Cannot handle `BLOB` or `TEXT` columns or updates (yet).

We update most of our production databases with the Unireg interface and serve web pages through **MySQL** (and in some extreme cases the Unireg report generator).

Unireg takes about 3M of disk space and works on at least the following platforms: SunOS 4.x, Solaris, Linux, HP-UX, ICL Unix, DNIX, SCO and MS-DOS.

Unireg is currently only available in Swedish and Finnish.

The price tag for Unireg is 10,000 Swedish kr (about $1500 US), but this includes support. Unireg is distributed as a binary. (But all the ISAM sources can be found in **MySQL**). Usually we compile the binary for the customer at their site.

All new development is concentrated to **MySQL**.

Appendix J The MySQL server license for non Microsoft operating systems

MySQL FREE PUBLIC LICENSE

(Version 4, March 5, 1995)

NOTE: This license is not the same as any of the GNU Licenses published by the Free Software Foundation. Its terms are substantially different from those of the GNU Licenses. If you are familiar with the GNU Licenses, please read this license with extra care.

This License applies to the computer program known as "MySQL". The "Program", below, refers to such program, and a "work based on the Program" means either the Program or any derivative work of the Program, as defined in the United States Copyright Act of 1976, such as a translation or a modification. The Program is a copyrighted work whose copyright is held by TcX Datakonsult AB and Monty Program KB and Detron HB.

This License does not apply when running "MySQL" on any Microsoft operating system. Microsoft operating systems include all versions of Microsoft Windows NT and Microsoft Windows.

BY MODIFYING OR DISTRIBUTING THE PROGRAM (OR ANY WORK BASED ON THE PROGRAM), YOU INDICATE YOUR ACCEPTANCE OF THIS LICENSE TO DO SO, AND ALL ITS TERMS AND CONDITIONS FOR COPYING, DISTRIBUTING OR MODIFYING THE PROGRAM OR WORKS BASED ON IT. NOTHING OTHER THAN THIS LICENSE GRANTS YOU PERMISSION TO MODIFY OR DISTRIBUTE THE PROGRAM OR ITS DERIVATIVE WORKS. THESE ACTIONS ARE PROHIBITED BY LAW. IF YOU DO NOT ACCEPT THESE TERMS AND CONDITIONS, DO NOT MODIFY OR DISTRIBUTE THE PROGRAM.

1. Licenses.

   Licensor hereby grants you the following rights, provided that you comply with all of the restrictions set forth in this License and provided, further, that you distribute an unmodified copy of this License with the Program:

   a. You may copy and distribute literal (i.e., verbatim) copies of the Program's source code as you receive it throughout the world, in any medium.

   b. You may modify the Program, create works based on the Program and distribute copies of such throughout the world, in any medium.

2. Restrictions.

   This license is subject to the following restrictions:

   a. Distribution of the Program or any work based on the Program by a commercial organization to any third party is prohibited if any payment is made in connection with such distribution, whether directly (as in payment for a copy of the Program) or indirectly (as in payment for some service related to the Program, or payment for some product or service that includes a copy of the Program "without charge"; these are only examples, and not an exhaustive enumeration of prohibited activities). However, the following methods of distribution involving payment shall not in and of themselves be a violation of this restriction:

A. Posting the Program on a public access information storage and retrieval service for which a fee is received for retrieving information (such as an on-line service), provided that the fee is not content-dependent (i.e., the fee would be the same for retrieving the same volume of information consisting of random data).

B. Distributing the Program on a CD-ROM, provided that the files containing the Program are reproduced entirely and verbatim on such CD-ROM, and provided further that all information on such CD-ROM be redistributable for non-commercial purposes without charge.

b. Activities other than copying, distribution and modification of the Program are not subject to this License and they are outside its scope. Functional use (running) of the Program is not restricted, and any output produced through the use of the Program is subject to this license only if its contents constitute a work based on the Program (independent of having been made by running the Program).

c. You must meet all of the following conditions with respect to the distribution of any work based on the Program:

A. If you have modified the Program, you must cause your work to carry prominent notices stating that you have modified the Program's files and the date of any change;

B. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole and at no charge to all third parties under the terms of this License;

C. If the modified program normally reads commands interactively when run, you must cause it, at each time the modified program commences operation, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty). Such notice must also state that users may redistribute the Program only under the conditions of this License and tell the user how to view the copy of this License included with the Program. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.);

D. You must accompany any such work based on the Program with the complete corresponding machine-readable source code, delivered on a medium customarily used for software interchange. The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable code. However, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable code;

E. If you distribute any written or printed material at all with the Program or

any work based on the Program, such material must include either a written copy of this License, or a prominent written indication that the Program or the work based on the Program is covered by this License and written instructions for printing and/or displaying the copy of the License on the distribution medium;

F. You may not impose any further restrictions on the recipient's exercise of the rights granted herein.

If distribution of executable or object code is made by offering the equivalent ability to copy from a designated place, then offering equivalent ability to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source code along with the object code.

3. Reservation of Rights.

No rights are granted to the Program except as expressly set forth herein. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

4. Other Restrictions.

If the distribution and/or use of the Program is restricted in certain countries for any reason, Licensor may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

5. Limitations.

THE PROGRAM IS PROVIDED TO YOU "AS IS," WITHOUT WARRANTY. THERE IS NO WARRANTY FOR THE PROGRAM, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL LICENSOR, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Appendix K The MySQL license for Microsoft operating systems

MySQL shareware license for Microsoft operating systems
(Version 1, September 4, 1998)

This License applies to the computer program known as "MySQL".

This License applies when running MySQL on any Microsoft operating system. Microsoft operating systems include all versions of Microsoft Windows NT and Microsoft Windows.

YOU SHOULD CAREFULLY READ THE FOLLOWING TERMS AND CONDITIONS BEFORE USING, COPYING OR DISTRIBUTING MySQL. BY USING, COPYING AND DISTRIBUTING MySQL, YOU INDICATE YOUR ACCEPTANCE OF THIS LICENSE TO DO SO, AND ALL ITS TERMS AND CONDITIONS FOR USING, COPYING AND DISTRIBUTING MySQL OR WORKS BASED ON IT. NOTHING OTHER THAN THIS LICENSE GRANTS YOU PERMISSION TO USE, COPY OR DISTRIBUTE MySQL OR ITS DERIVATIVE WORKS. THESE ACTIONS ARE PROHIBITED BY LAW. IF YOU DO NOT ACCEPT THESE TERMS AND CONDITIONS, DO NOT USE, COPY OR DISTRIBUTE MySQL.

1. Evaluation and License Registration.

   This is an evaluation version of MySQL for Win32. Subject to the terms below, you are hereby licensed to use MySQL for evaluation purposes without charge for a period of 30 days. If you use MySQL after the 30 day evaluation period the registration and purchase of a MySQL license is required.

   The price for a MySQL license is currently 200 US dollars and email support starts from 200 US dollars/year. Quantity discounts are available. If you pay by credit card, the currency is EURO (The European Unions common currency) so the prices will differ slightly.

   The easiest way to register or find options about how to pay for MySQL is to use the license form at TcX's secure server at https://www.mysql.com/license.htmy. This can be used also when paying with credit card over the Internet.

   Other applicable methods for paying are SWIFT payments, cheques and credit cards.

   Payment should be made to:

   ```
   Postgirot Bank AB
   105 06 STOCKHOLM, SWEDEN

   TCX DataKonsult AB
   BOX 6434
   11382 STOCKHOLM, SWEDEN

   SWIFT address: PGSI SESS
   Account number: 96 77 06 - 3
   ```

   Specify: license and/or support and your name and email address.

In Europe and Japan, EuroGiro (that should be cheaper) can be used to the same account.

If you want to pay by cheque make it payable to "Monty Program KB" and mail it to the address below.

```
TCX DataKonsult AB
BOX 6434
11382 STOCKHOLM, SWEDEN
```

For more information about commercial licensing, please contact:

```
David Axmark
Kungsgatan 65 B
753 21 UPPSALA
SWEDEN
Voice Phone +46-18-10 22 80     GMT 9-21. Swedish and English spoken
Fax +46-8-729 69 05             Email *much* preferred.
E-Mail: mysql-licensing@mysql.com
```

For more about the license prices and commercial support, like email support, please refer to the MySQL manual. See Section 3.5 [Cost], page 24. See Section 3.6 [Support], page 26.

The use of MySQL or any work based on MySQL after the 30-day evaluation period is in violation of international copyright laws.

2. Registered version of MySQL.

   After you have purchased a MySQL license we will send you a receipt by paper mail. You are allowed to use MySQL or any work based on MySQL after the 30-days evaluation period. The use of MySQL is, however, restricted to one physical computer, but there are no restrictions on concurrent uses of MySQL or the number of MySQL servers run on the computer.

   We will also email you an address and password for a password-protected WWW page that always has the newest MySQL-Win32 version. Our current policy is that a user with the MySQL license can get free upgrades. The best way to ensure that you get the best possible support is to purchase commercial support!

3. Registration for use in education and university or government-sponsored research.

   You may obtain a MySQL license for the use in education and university or government-sponsored research for free. In that case, send a detailed application for licensing MySQL for such use to the email address `mysql-licensing@mysql.com`.

   The following information is required in the application:

   - The name of the school or institute.
   - A short description of the school or institute and of the type of education, resarch or other functions it provides.
   - A detailed report of the use of MySQL in the institution.

   In this case you will be provided with a license that entitles you to use MySQL in a specified manner.

4. Distribution.

   Provided that you verify that you are distributing an evaluation or educational/research version of MySQL you are hereby licensed to make as many literal (i.e., verbatim) copies of the evaluation version of MySQL and documentation as you wish.

5. Restrictions.

   The client code of MySQL is in the Public Domain or under the GPL (for example the code for readline) license. You are not allowed to modify, recompile, translate or create derivative works based upon any part of the server code of MySQL.

6. Reservation of Rights.

   No rights are granted to MySQL except as expressly set forth herein. You may not copy or distribute MySQL except as expressly provided under this License. Any attempt otherwise to copy or distribute MySQL is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

7. Other Restrictions.

   If the distribution and/or use of MySQL is restricted in certain countries for any reason, the Licensor may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

8. Limitations.

   MySQL IS PROVIDED TO YOU "AS IS," WITHOUT WARRANTY. THERE IS NO WARRANTY FOR MySQL, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF MySQL IS WITH YOU. SHOULD MySQL PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

   IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL THE LICENSOR, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE MySQL AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE MySQL (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF MySQL TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

SQL command, type and function index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Concept Index

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Short Contents

Table of Contents