CHAPTER 1

What Is MySQL?

THIS CHAPTER BEGINS WITH AN overview of the most important concepts from the world of databases and then delves into the possibilities and limitations of MySQL. What is MySQL? What can it do (and what is it unable to do)?

In addition to describing the central functions of MySQL we shall also discuss fully the issue of licensing MySQL. When is one permitted to use MySQL without payment, and when is a license required?

Chapter Overview

What Is a Database?

Before we can answer the central question of this chapter, namely, *What is MySQL?* you, dear reader, and I must find a common language. Therefore, this section presents a beginning database glossary, without going into great detail. (If you have already had significant dealings with relational databases, you can skip the next couple of pages in good conscience.)

There is scarcely to be found a term that is less precise than *database.* A database can be a list of addresses residing in a spreadsheet program (such as Excel), or it can be the administration files of a telecommunications firm in which several million calls are registered daily, their charges accurately calculated, monthly bills computed, and warning letters sent to those who are in arrears. A simple database can be a stand-alone operation (residing locally on a computer for a single user), while others may be used simultaneously by thousands of users, with the data parceled out among several computers and dozens of hard drives. The size of a database can range from a few of kilobytes into the terabytes.[1]

In ordinary usage the word "database" is used to refer to the actual data, the resulting database files, the database system (such as MySQL or Oracle), or a database client (such as a PHP script or a program written in C++). Thus there arises a great potential for confusion as soon as two people begin to converse on the subject of databases.

Relations, Database Systems, Servers, and Clients

A *database* is an ordered collection of data, which is normally stored in one or more associated files. The data are structured as *tables*, where cross references among tables are possible. The existence of such *relations* among the tables leads to the database being called a *relational database.*

Let us clarify matters with an example. A database might consist of a table with data on a firm's customers (name, address, etc.), a table with data on the products the firm offers, and finally, a table containing the firm's orders. Through the table of orders it is possible to access the data in the other two tables (for example, via customer and product numbers).

MySQL, Oracle, the Microsoft SQL server, and IBM DB2 are examples of *relational database systems.* Such a system includes the programs for managing relational databases. Among the tasks of a relational database system are not only the secure storage of data, but also such jobs as the processing of commands for

[1] It all started with the megabyte, which is about one million bytes. A terabyte is 1024 gigabytes, which in turn is approximately one thousand megabytes. The prefix "mega-" comes from the Greek for "great," or "large," while "giga-" is derived from the Greek word for "giant." In turn, "tera-" is from the Greek word for "monster." It would appear that numbers once regarded as large, gigantic, or even monstrously huge have become part of our everyday vocabulary.

querying, analyzing, and sorting existing data and for storing new data. All of this should be able to take place not only on a single computer, but over a network as well. Instead of a *database system* we shall often speak of a *database server*.

Where there are servers, there are clients. Every program that is connected to the database system is called a *database client*. Database clients have the job of simplifying the use of the database for the end user. No user of a database system in his or her right mind would wish to communicate directly with the database server. That is much too abstract and inconvenient. (Let programmers worry about such direct communication!) Instead, the user has a right to expect convenient tables, list boxes, and so on to enable the location of data or to input new data.

Database clients can assume a variety of forms, and indeed, they are often not recognized by the user as database programs at all. Some examples of this type of client are HTML pages for the display and input of messages in an on-line discussion group, a traditional program with several windows for managing addresses and appointments, and a Perl script for executing administrative tasks. There is thus wide scope for database programming.

Relational Versus Object-Oriented Database Systems

Relational databases have dominated the database world for decades, and they are particularly well suited for business data, which usually lend themselves to structuring in the form of tables. Except for the following two paragraphs, this entire book discusses only relational databases (though we shall not always stress this point).

Another kind of database is the *object-oriented database*. Such databases can store free-standing objects (without having to arrange them in tables). Although in recent years there has been a trend in the direction of object-oriented programming languages (such as Object-Store, O2, Versant, Objectivity), object-oriented databases have found only a small market niche.

Note that relational databases can be accessed by means of object-oriented programming languages. However, that does not turn a relational database into an object-oriented one. Object-oriented database systems enable the direct access to objects defined in the programming language in question and the storage of such objects in the database without conversion (persistency). It is precisely this that is not possible with relational database systems, in which everything must be structured in tables.

Tables, Records, Fields, Queries, SQL, Index, Keys

We have already mentioned tables, which are the structures in which the actual data are located. Every line in such a table is called a *data record*, or simply *record*, where the structure of each record is determined by the definition of the table. For example, in a table of addresses every record might contain *fields* for family name, given name, street, and so on. For every field there are precise conditions on the type of information that can be stored (such as a number in a particular format, or a character string with a predetermined maximum number of characters). The collection of all of one type of field taken over all the rows of a table can be thought of as a *column* of the table, and we will sometimes be a bit loose in distinguishing fields from columns.

The description of a database consisting of several tables with all of its fields, relations, and indexes (see below) is called a *database model*. This model defines the construction of the data structures and at the same time provides the format in which the actual data are to be stored.

Tables usually contain their data in no particular order (more precisely, the order is usually that in which the data have been entered or modified). However, for efficient use of the data it is necessary that from these unordered data a list can be created that is ordered according to one or more criteria. It is frequently useful for such a list to contain only a selection of the data in the table. For example, one could obtain a list of all of one's customers, ordered by ZIP code, who have ordered a rubber ducky within the past twelve months.

To create such a list one formulates *queries*. The result of the query is again a table, one, however, that exists in active memory (RAM) and not on the hard drive.

To formulate a query one uses SQL instructions, which are commands for selecting and extracting data. The abbreviation SQL stands for *Structured Query Language*, which has become a standard in the formulation of database queries. Needless to say, every producer of a database system offers certain extensions to this standard, which dilutes the goal of compatibility among various database systems.

When tables get large, the speed at which a query can be answered depends significantly on whether there is a suitable *index* giving the order of the data fields. An index is an auxiliary table that contains only information about the order of the records. An index is also called a *key*.

An index speeds up access to data, but it has disadvantages as well. First, every index increases the amount of storage on the hard drive necessary for the database file, and second, the index must be updated each time the data are altered, and this costs time. (Thus an index saves time in the reading of data, but it costs time in entering and altering data. It thus depends on the use to which the data are to be put whether an index is on the whole a net plus or minus in the quest for efficiency.)

A special case of an index is a *primary index*, or *primary key*, which is distinguished in that the primary index must ensure a *unique* reference to a record. Often, for this purpose one simply uses a running index number (ID number). Primary indexes play a significant role in relational databases, and they can speed up access to data considerably.

MySQL

MySQL is a relational database system. If you can believe many diehard MySQL fans, MySQL is faster, more reliable, and cheaper—or, simply put, better—than any other database system (including commercial systems such as Oracle and DB2). Many MySQL opponents challenge this viewpoint, going even so far as to assert that MySQL is not even a relational database system. We can safely say that there is a large bandwidth of opinion.

- The fact is that there is an ever increasing number of MySQL users, and the overwhelming majority of them are quite satisfied with MySQL. Thus for these users we may say that MySQL is *good enough*.
- It is also the fact, however, that MySQL lacks many features that are taken for granted with other database systems. If you require such features, then MySQL is (at least for the present) not the database system for you. MySQL is not a panacea.

In the next couple of sections we shall examine some of the possibilities and limitations of MySQL.

> **REMARK** *In this book we are considering MySQL. Do not confuse MySQL with mSQL. To be sure, MySQL and mSQL have similar programming interfaces (APIs), which merely facilitates the confusion between the two. However, the underlying database systems are completely different. More information on mSQL can be found at* `http://www.hughes.com.au/`

Features of MySQL

- **Relational Database System:** Like almost all other database systems on the market, MySQL is a relational database system.

- **Client/Server Architecture:** MySQL is a client/server system. There is a database server (MySQL) and arbitrarily many clients (application programs), which communicate with the server; that is, they query data, save changes, etc. The clients can run on the same computer as the server or on another computer (communication via a local network or the Internet).

  Almost all of the familiar large database systems (Oracle, Microsoft SQL Server, etc.) are client/server systems. These are in contrast to the *file-server systems*, which include Microsoft Access, dBase, and FoxPro. The decisive drawback to file-server systems is that when run over a network they become extremely inefficient as the number of users grows.

- **SQL:** MySQL supports as its database language—as its name suggests—SQL (Structured Query Language). SQL is a standardized language for querying and updating data and for the administration of a database.

  There are several SQL dialects (about as many as there are database systems). MySQL adheres to the ANSI-SQL/92 standard, although with some significant restrictions and many extensions.

  This topic will be dealt with more extensively below. Beyond the ANSI-SQL/92 standard, MySQL supports, among other things, several additional data types, full-text indexes, and replication.

- **Programming Languages:** There is a host of APIs (application programming interfaces) and libraries for the development of MySQL applications. For client programming you can use, among others, the languages C, C++, Java, Perl, PHP, Python, and Tcl.

- **ODBC:** There is an ODBC interface for MySQL. With it MySQL can be addressed by all the usual programming languages running under Microsoft Windows (Delphi, Visual Basic, etc.). MyODBC is currently at version 2.5, level 0. The ODBC interface can also be installed under Unix, though that is seldom necessary.

- **Platform Independence:** It is not only client applications that can run under various operating systems. MySQL itself (that is, the server) can also be run under a variety of operating systems. The most significant are Apple Macintosh OS X, IBM OS/2, Linux, Microsoft Windows, as well as countless flavors of Unix (such as AIX, BSDI, DEC Unix, FreeBSD, HP-UX, Open-BSD, Net BSD, SGI Iris, Sun Solaris, SunOS 4, SCO Unix).

- **Speed:** MySQL is considered a fast database system. This assessment has been supported by countless benchmark tests (though such tests, regardless of who has carried them out, should be regarded with a certain degree of skepticism). In part, MySQL's speed advantage is the result of the absence of certain features.

Limitations

This section is aimed at the reader who already has some knowledge of relational databases. We shall use some terms from the world of databases without defining them precisely. (On the topic of transactions alone one could fill dozens of pages!)

However, our explanations should be understandable to the database novice as well. If you are interested in more detail on such background material, then I must refer you to the vast literature (see the Bibliography at the end of the book).

Many of the shortcomings listed in this section can be found on the to-do list of the team of MySQL developers, or have already been implemented. But even if one or another feature is already available by the time this book appears, this does not mean that you can use it in your applications without further thought. The development of database systems, whether by a commercial firm or in the open-source arena, is a very complicated endeavor. So wait until features have been thoroughly tested and have had a chance to mature.

> **POINTER** *The documentation for MySQL is not at all silent on the subject of shortcomings or missing features. There is a quite readable chapter in the MySQL documentation on the topic "How standards-compatible is MySQL?" There you will find extensive information on the points at which MySQL fails to comply with current standards. Often, a reason for the shortcoming is provided, and sometimes as well some pointers on how to get around the difficulty.*

ANSI-SQL/92 Limitations

ANSI SQL/92 is a standardized definition of the database query language SQL. Many commercial database systems are largely compatible with this standard and also offer many extensions.

MySQL is also conspicuous for its countless extensions, but unfortunately, the compatibility is not so extensive as is the case for other database systems. The points described in greater detail in what follows (transactions, sub*SELECT*s, views, triggers, stored procedures, foreign keys) are examples of the lack of ANSI-SQL/92 compatibility.

These limitations usually make it impossible (or merely very difficult) to adapt existing databases and the associated SQL code from other database systems into MySQL. Conversely, it is almost as difficult to transfer a MySQL solution to another database system if you have not scrupulously employed only the ANSI-SQL/92-conforming features of MySQL. (The temptation to use the useful proprietary features proves in most cases to be too great.)

- **SubSELECTs:** MySQL is not capable of executing a query of the form

  ```
  SELECT * FROM table1 WHERE x IN (SELECT y FROM table2)
  ```

  This limitation can be circumvented in many cases by setting up a temporary table, which, however, is neither elegant nor particularly efficient. In other cases the limitation must be attacked with additional code in an external programming language.

  There are similar limitations for *DELETE*.

- **Foreign Keys:** MySQL is conversant with *foreign keys*, which help to link two tables. Usually, however, the keyword *foreign keys* also describes the ability of a database to ensure the referential integrity of the linked tables, and at present MySQL is incapable of this.

  Let us suppose that we have two tables, one for book titles and a second for the authors of these books. If you delete an author from the author table, the database system should test whether any book titles by this author remain in the database. If that is the case, then either these titles should be deleted as well or the entire operation should be aborted.

  MySQL is not concerned with such relations. If you delete the author, the remaining book records refer to a no-longer-existing author. (A similar situation obtains on the Internet, where a link all too frequently points to a nonexistent site). Not only is this situation unaesthetic, it often causes serious problems in programming and debugging.

  Therefore, in MySQL database applications the programmer must ensure that the integrity of the data (that is, the relationships among the various tables) is maintained when commands for changing and deleting data are executed.

- **Views:** Simply put, with *views* we are dealing with an SQL query that is considered an independent database object and that permits a particular view into the database. MySQL currently does not support views (though we may expect this feature to be implemented shortly).

  Of course, with MySQL you can always execute those SQL commands that are defined as views in other databases. However, it is impossible to store these queries as objects in the database.

  Views assist in the administration of the database, since it is thereby relatively simple to control access to individual *parts* of the database. Views also help in avoiding redundancy in application development.

- **Stored Procedure:** When we speak of *stored procedures* we are referring to SQL code that is stored within a database system. Stored procedures (SPs for short) are usually employed to simplify certain steps in a procedure,

such as inserting or deleting a record. For client programmers this has the advantage that these actions need not operate directly on the tables, but can rely on SPs. As with views, SPs also help in the administration of large database projects. Stored procedures can also increase efficiency in certain cases.

- **Triggers:** A *trigger* is an SQL command that is automatically executed by the server during certain database operations. If a database is incapable on its own of ensuring referential integrity, then it is usually triggers that are brought into play to assist in this. For example, a trigger can be executed every time a record is to be deleted, which would test whether this operation is permissible and prohibit the action if need be.

- **Transactions:** A *transaction* in the context of a database system refers to the execution of several database operations as a *block*, that is, as if it were a single command. The database system ensures that either all of the operations are properly executed or else none of them. This holds even if during the transaction a power interruption occurs, the computer crashes, or some other catastrophe intervenes. Thus, for example, it cannot happen that $100,000 is withdrawn from your bank account without then being deposited into mine if an error intervenes, no matter of what kind.

  Transactions also give programmers the option of effectively aborting any number of already executed SQL commands. In many situations this simplifies programming considerably.

  There is a range of opinion as to the importance of transactions. The MySQL development team has long held the belief that transactions are not necessary in many applications and that MySQL runs sufficiently securely and stably without them. Furthermore, avoiding transactions yields considerable savings in speed. And finally, every programmer has the option of writing code that ensures that a series of SQL commands terminates successfully or, if not, can be properly withdrawn. (This argument, however, sounds almost as if, say, one were to suggest to C++ programmers that they deal with memory management of objects themselves.)

  Be that as it may, MySQL has supported transactions since May 2000 in the form of BDB tables. and also since April 2001 in the form of InnoDB tables. A third type of what can be considered a transaction, Gemini, should arrive shortly. However, the MySQL documentation admits that these new table types have not yet been tested and checked to the same extent as other MySQL features. A further problem is that these new table types are not yet available for all operating systems that support MySQL. In other words, MySQL supports transactions, but this feature will take a while to mature and become stable. See Chapter 12 for more on transactions.

The lack of sub*SELECT*s, views, SPs, and triggers does not greatly restrict the possibilities open to client programmers. It does, however, lead to a situation in which the program logic is transferred from the server level to the client level. The result is more complex or expensive client programming than would otherwise be the case, leading to redundancies in code and problems with maintenance and alteration of code.

Further Limitations

The following limitations actually have nothing to do with ANSI-SQL/92 but nonetheless play a significant role in practice.

- When MySQL is used with standard tables (table type MyISAM), *locking*—that is, the temporary blocking of access to or alteration of database information—is in operation only for entire tables (*table locking*). With other database systems, on the other hand, it is possible to protect only certain records using locking (*row locking*). This is much more efficient when several users are accessing the database at the same time. In many database applications the difference in speed obtained from better locking is so great that MySQL is ruled out as a database system.

  You can circumvent the *table-locking* problem in MySQL by implementing BDB tables. Transactions in BDB tables are executed without *table locking* and are therefore executed more efficiently than they would be with *LOCK*-protected operations in normal MySQL tables.

- MySQL is not able to execute *hot backups,* which are backups during operation without blocking the tables with locks.

- The MySQL server supports a variety of character sets and the associated sort orders, but at present can have only a single *sort order* active at a time. If the sort order must be changed, MySQL has to be restarted. All table indexes must then be reset.

  Especially for those using MySQL with Internet service providers it would make sense if the desired sort order could be set individually for each table or for each client.

- MySQL is currently unable to deal with Unicode or other mutibyte character strings. (Of course, it is possible to treat such data as binary objects, but there are no functions for dealing with character strings, in particular no usable sorting or comparison algorithms.)

- A *development environment* (front end, GUI) for administrative tasks—from database development to backup—has become a given with many commercial database systems. With MySQL there is not merely a

development environment, there is a whole dozen of them. To be sure, tools such as phpMyAdmin are already fairly mature and represent a significant alleviation of administrative work, but there is presently no tool that covers the complete spectrum of all administrative tasks in a suitable and user-friendly manner.

MySQL 4.0/4.1

As this book was being finished, MySQL version 4.0 had just been announced but was not yet available, while version 4.1 is most likely still a gleam in the development team's eye.

MySQL 4.0

Despite its shiny new version number, MySQL 4.0 is not set to offer many new functions, but instead, to serve as the basis for future extensions. The following list provides a brief preview of the most important changes to look for.

- The new table types—either already available or soon to be—InnoDB and Gemini should be completely mature by version 4.0. Then MySQL will boast three transaction-capable table types, namely BDB, InnoDB, and Gemini. Additional features of InnoDB and Gemini are support of row-level locking and automatic reconstruction of tables after a crash. Additional details on these new table types can be found at `www.innobase.fi` and `www.nusphere.com`.
- It should be possible to execute hot backups while the system is running (without blocking the entire database).
- The format of files for describing table properties (`*.frm`) is set to change.
- The MySQL server `mysqld` should be available not only as a program, but also as a library, so that MySQL can be better used in *embedded products*.

POINTER *Information on the current status of MySQL development can be found at* `http://www.mysql.com/development/`

MySQL 4.1

Version 4.1 had not even been announced as this book was being written. But according to the MySQL mailing list, this version will contain, among other things, the long-awaited sub*SELECT* commands.

MySQL Licensing

One of the most interesting features of MySQL is the license. MySQL is an open source project. That is, the complete source code of MySQL is freely available. Since June 2000 (that is, since version 3.23.19) the *GNU Public License* (GPL) is valid also for MySQL. It is thus ensured that MySQL will continue to be freely available in the sense of the open source idea. (For commercial applications of MySQL there is a second, commercial, license available in addition to GPL. More on this later.)

Rights and Duties with Respect to the GPL

Open source is often incorrectly interpreted to mean "without cost." It is indeed true that GPL software can be used without payment of fees, provided that one adheres to certain conditions. However, the open source idea goes much further:

- Since the source code is freely available, when there are problems you are not at the mercy of a software vendor.
- When problems arise you can attempt to repair the problem yourself or to implement features that are lacking. Furthermore, you can appeal to the developers' group for help.
- You can be certain that the program code has been read by many developers and does not contain any unsavory surprises (such as so-called back doors such as the database system Interbase had for many years, whereby access to every Interbase database was possible via a hard-coded password).
- You are permitted to alter GPL products, and indeed sell the resulting new programs.

At the end of this list of GPL merits there are a few demerits (for commercial applications). If you wish to use a GPL program as the basis for a commercial product, you must again make your own source code freely available, in the sense of GPL, with the changes made. This is seldom something that developers of commercial products wish to do.

In general, it holds that every program that is derived from GPL software exists under the terms of GPL. Since this feature would often too severely restrict

the implementation of GPL programs, there is an additional form of GPL license: In addition to the ordinary GPL, whose rules we have just gone over, there is the LGPL (Library GPL). The LGPL allows for the free use of LGPL software in commercial applications without the commercial code having to be made available in the sense of GPL. (In the case of MySQL, MySQL servers and the administration tools adhere to the terms of GPL, while the MySQL client libraries adhere to LGPL.)

> **POINTER** *Further information on the open source idea, including the full text of the GPL together with aids to interpretation, can be found at the following addresses:*
>
> `http://www.gnu.org/copyleft/gpl.html`
> `http://www.opensource.org/osd.html`

MySQL Licensing in Practice

MySQL adheres not only to GPL, but also to a second, commercial, license. According to the application, you can choose between the open source license and a commercial license.

Use of MySQL in the Sense of the Open Source License

The following list collects the different situations in which one may freely use MySQL in the sense of GPL.

- MySQL can always be used internally without cost. It is only when the resulting solution is to be sold to other customers that the question of licensing comes into play.
- MySQL can be used freely within a web site. (The code of PHP or Perl scripts need not be made freely available in the sense of GPL, since you, as a PHP or Perl developer, communicate with MySQL via the MySQL client library, which is subordinate to LGPKL.)
- Likewise, an Internet service provider may make MySQL available to its customers without having to pay MySQL license fees. (Since MySQL is running exclusively on the ISP computer, this application is considered internal.)

- Finally, MySQL can be used free of charge for all projects that themselves run under the GPL or comparable free license. (If you have developed a new free e-mail client for Linux, say, and wish to store e-mails in a MySQL database, you may do so without further ado.)

Use of MySQL with a Commercial License

In the sense of GPL the following uses are prohibited:

- You may not change or extend MySQL (that is, the database server) itself or sell the new version or product thus created without simultaneously making the source code of your changes freely available. You are thus prohibited from developing a new database system based on MySQL if you are not prepared to make your extensions freely available to the MySQL community in the sense of GPL.
- It is forbidden to develop a commercial product, such as a bookkeeping program, for example (without making the code available in the open source sense), that relies exclusively on MySQL as the only possible database.

  The boundaries here are somewhat fluid. Thus according to the MySQL documentation itself it is permissible for a commercial program to provide an option for recording MySQL tables instead of logging files. However, if your program can be used only if MySQL is also available, thus creating a direct dependency on MySQL, then a commercial license is required.

If the limitations of the GPL are not acceptable to you as a commercial developer, then you have the right to apply for a commercial MySQL license. This can prove worthwhile because MySQL remains available to you even if you are unable or unwilling to make your code available in the sense of GPL.

Commercial MySQL licensing fees are calculated based on the number of servers (where a server is a computer, regardless of the number of CPUs). There is no limitation on the number of clients that may access the server. The cost is quite reasonable in comparison to commercial database systems (currently $200 for a license, with a significant reduction starting at ten licenses).

REMARK *If you use MySQL in combination with BDB, InnoDB, or Gemini tables (and not with the default table type MyISAM), the commercial license fee is increased by about thirty percent for each additional table type. The reason for this is the three table types have been developed and are maintained rather independently of MySQL.*

POINTER *Further information on licensing MySQL can be found in the MySQL documentation, in the chapter "MySQL Licensing and Support."*

Support Contracts

You may make a support contract with the MySQL company if you desire commercial support in the development of a MySQL application. You thereby simultaneously support the further development of MySQL.

POINTER *Details on commercial MySQL licenses and paid support can be found at the following address:*

```
http://www.mysql.com/support/
```

Links to various companies that offer commercial MySQL support can be found at the following address:

```
http://www.mysql.com/information/partners.html
```

Alternatives to MySQL

Of course, there are many alternatives to MySQL, particularly if you are prepared to pay (lots of) money for licenses and perhaps also for the requisite hardware. Among these are IBM DB2, Informix, Microsoft SQL Server, and Oracle.

If you are looking for a database in the open source realm, then PostgreSQL is currently perhaps the most interesting alternative. (However, be warned: The discussion between advocates of MySQL and those of PostgreSQL usually resembles more a war of religions than what might be termed measured intellectual discourse.)

Furthermore, there are several formerly commercial database systems that have recently been converted to open source. The two best known of these are Interbase (from Inprise/Borland) and SAP DB.

POINTER *Much further information on PostgreSQL can be found, needless to say, on the Internet. The following web sites are a good starting point:*

```
http://www.at.postgresql.org/
http://www.postgres.com
http://openacs.org/
```

POINTER *Comparisons between various database systems (including MySQL and PostgresSQL can be found at the following sites, among others:*

```
http://www.postgresql.org/mhonarc/pgsql-general/1999-11/msg00227.html
http://www.phpbuilder.com/columns/tim20001112.php3
http://www.devshed.com/BrainDump/MySQL_Benchmarks/
http://openacs.org/philosophy/why-not-mysql.html
```

POINTER *Three home pages on Interbase and open source projects derived therefrom and a page on SAP DB are the following:*

```
http://www.interbase2000.org
http://www.ibphoenix.com
http://firebird.sourceforge.net
http://www.sapdb.org
```

Summary

MySQL is a very capable relational client/server database system. It is sufficiently secure and stable for many applications, and it offers an excellent cost/benefit ratio (not only because MySQL is free itself, but also because it makes comparatively modest demands on hardware). MySQL has thus developed into a quasi standard in the realm of small-to-mid-sized Internet databases (such as for discussion groups or similar applications).

Above all, in the Linux world MySQL is used increasingly by applications as the background database engine, whether it be managing logging data more efficiently than previously or managing e-mail, MP3 files, addresses, or comparable data. MySQL is poised to play a similar role to that of the Jet Engine in the Microsoft operating system (where in many respects MySQL offers s meaningfully better technical basis). Thanks to the ODBC interface, MySQL is now being used in the Windows world for such tasks.

Apart from technical data, MySQL has the advantage over other open source database systems in that presently it is by far the most popular database system. It follows that MySQL has been more thoroughly tested and documented than other database systems and that it is relatively easy to find developers with MySQL experience.

However, MySQL cannot (yet) compete in every respect with the big boys of the commercial database system world. If you must manage highly critical data (to give two examples: on-line banking and management of medical data) or if you have special requirements (such as data warehousing), then MySQL if most likely not going to be your first choice.