Edgar F. Codd introduced the relational model to database management in 1970 (a copy of his seminal paper is available online at www.acm.org/classics/nov95). The model has since become a standard for database management systems (DBMSs). Relational databases store not only information about data items but also information about the relationships between them. Relational databases are powerful because they impose minimal constraints on the kinds of relationships they can represent and on how data is extracted from them. Popular relational DBMSs (RDBMSs) include commercial products, such as Oracle (www.oracle.com), Microsoft SQL Server (www.microsoft.com/sql), and Sybase (www.sybase.com), and open source products, such as PostgreSQL (www.postgresql.com) and MySQL (www.mysql.com). All of these products use the Structured Query Language for extracting database information.

In recent years, the open source products have achieved enterprise-level quality. In response, enterprises have become more interested in migrating from proprietary, commercial products to open source. Businesses around the world now commonly use the two leading open source relational DBMSs, MySQL and PostgreSQL.

The Los Alamos National Laboratory’s Research Library has used MySQL databases for years. However, a recent project to develop a comprehensive database of scientific journal articles and citation information revealed its unique strengths and features.

Digital library requirements

Digital libraries as well as data centers have demanding requirements for capacity, speed, reliability, and flexibility. Determining the best relational DBMS isn’t an easy task, and it depends on the criteria the application must satisfy. Key RDBMS capabilities include the following:

- Multiuser simultaneous access.
- Easy access from APIs written in different languages. The two most well-known APIs are Open Database Connectivity and Java Database Connectivity. An API provides connectivity to the database, and SQL statements manipulate it. The combination of APIs and SQL establishes nearly complete interoperability between the database and a client.
- Support for multiple operations that happen either all at once or not at all. A true trans-
We chose MySQL, which is developed, supported, and marketed by MySQL AB, a commercial company that builds its business providing services for the database product. MySQL is most commonly used for Web and embedded applications. Its speed and reliability have made it a popular alternative to proprietary database systems. It runs on several platforms and has many attractive features. We chose it for the following reasons:

- It's easy to use, install, and maintain. It's well documented, has good support through the users’ group, and also offers commercial support.
- It provides fault tolerance, load balancing, and security via replication. Replication is not a backup policy, but it provides a basic level of protection against hardware failure. We also use it to update a server inside the firewall and propagate the data to servers outside the firewall in read-only mode.
- MySQL was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Its connectivity, speed, and security make MySQL server highly suited for accessing databases on the Internet.

Your requirements relative to key capabilities will determine which product works best for any given project. As a database architect and administrator, I'm concerned with all these key capabilities as well as the following:

- commercial support of at least modest usefulness, and
- a robust and complete logging mechanism for abnormal end conditions.

The decision must ultimately weigh not only performance, features, and support but also licensing and price factors. When cost is an issue, open source products offer free or inexpensive alternatives. We chose MySQL, which is developed, supported, and marketed by MySQL AB, a commercial company that builds its business providing services for the database product. MySQL is most commonly used for Web and embedded applications. Its speed and reliability have made it a popular alternative to proprietary database systems. It runs on several platforms and has many attractive features. We chose it for the following reasons:

- It offers several storage models, such as InnoDB, MyISAM, and FullText. InnoDB provides MySQL with a transaction-safe (ACID-compliant) storage engine that includes commit, rollback, and crash-recovery capabilities. It also implements locking on the row level and provides a consistent nonlocking read in SELECT statements. MyISAM supports storage of more than a terabyte of data in more than 200 tables in a single project database.
- It's fast. MySQL can manage links among our 1.5 billions rows of data in several virtual tables. It can handle hundreds of clients connecting to the server and using multiple databases simultaneously.
- It's easy to use, install, and maintain. It's well documented, has good support through the users’ group, and also offers commercial support.
- It provides fault tolerance, load balancing, and security via replication. Replication is not a backup policy, but it provides a basic level of protection against hardware failure. We also use it to update a server inside the firewall and propagate the data to servers outside the firewall in read-only mode.
- Replication support that scales and achieves high availability.
- It's fast. MySQL can manage links among our 1.5 billions rows of data in several virtual tables. It can handle hundreds of clients connecting to the server and using multiple databases simultaneously.
- It's fast. MySQL can manage links among our 1.5 billions rows of data in several virtual tables. It can handle hundreds of clients connecting to the server and using multiple databases simultaneously.

Your requirements relative to key capabilities will determine which product works best for any given project. As a database architect and administrator, I’m concerned with all these key capabilities as well as the following:

- commercial support of at least modest usefulness, and
- a robust and complete logging mechanism for abnormal end conditions.

The decision must ultimately weigh not only performance, features, and support but also licensing and price factors. When cost is an issue, open source products offer free or inexpensive alternatives. We chose MySQL, which is developed, supported, and marketed by MySQL AB, a commercial company that builds its business providing services for the database product. MySQL is most commonly used for Web and embedded applications. Its speed and reliability have made it a popular alternative to proprietary database systems. It runs on several platforms and has many attractive features. We chose it for the following reasons:

- It offers several storage models, such as InnoDB, MyISAM, and FullText. InnoDB provides MySQL with a transaction-safe (ACID-compliant) storage engine that includes commit, rollback, and crash-recovery capabilities. It also implements locking on the row level and provides a consistent nonlocking read in SELECT statements. MyISAM supports storage of more than a terabyte of data in more than 200 tables in a single project database.
- It's fast. MySQL can manage links among our 1.5 billions rows of data in several virtual tables. It can handle hundreds of clients connecting to the server and using multiple databases simultaneously.
- Replication support that scales and achieves high availability.
- It's easy to use, install, and maintain. It's well documented, has good support through the users’ group, and also offers commercial support.
- It's fast. MySQL can manage links among our 1.5 billions rows of data in several virtual tables. It can handle hundreds of clients connecting to the server and using multiple databases simultaneously.

Your requirements relative to key capabilities will determine which product works best for any given project. As a database architect and administrator, I’m concerned with all these key capabilities as well as the following:

- commercial support of at least modest usefulness, and
- a robust and complete logging mechanism for abnormal end conditions.

The decision must ultimately weigh not only performance, features, and support but also licensing and price factors. When cost is an issue, open source products offer free or inexpensive alternatives. We chose MySQL, which is developed, supported, and marketed by MySQL AB, a commercial company that builds its business providing services for the database product. MySQL is most commonly used for Web and embedded applications. Its speed and reliability have made it a popular alternative to proprietary database systems. It runs on several platforms and has many attractive features. We chose it for the following reasons:

- It offers several storage models, such as InnoDB, MyISAM, and FullText. InnoDB provides MySQL with a transaction-safe (ACID-compliant) storage engine that includes commit, rollback, and crash-recovery capabilities. It also implements locking on the row level and provides a consistent nonlocking read in SELECT statements. MyISAM supports storage of more than a terabyte of data in more than 200 tables in a single project database.
- It's fast. MySQL can manage links among our 1.5 billions rows of data in several virtual tables. It can handle hundreds of clients connecting to the server and using multiple databases simultaneously.
- Replication support that scales and achieves high availability.
- It's easy to use, install, and maintain. It's well documented, has good support through the users’ group, and also offers commercial support.

MySQL was originally developed to handle large databases much faster than existing solutions and has been successfully used in highly demanding production environments for several years. Its connectivity, speed, and security make MySQL server highly suited for accessing databases on the Internet.

Lessons learned

The LANL Research Library’s recent project to develop a comprehensive database of scientific journal articles and citation information was its most ambitious project ever. The project involved converting bibliographic metadata from several data sources into a common format and enhanced the data with links between each of more than 55 million articles as well as 600 million individual references. The project also provided search capabilities and browser access to the data.

In addition to the quantity of data to be managed, other challenges included maintaining flexibility, response time, reliability, fault tolerance, budgets, and security.

Replication

Because of network latency, it’s important to keep servers involved in the replication “closed.” Moreover, although MySQL replication works well, the process can break in cases of network outages, exhausted disk space, and other problems. It's therefore critical to monitor server’s status and error logs.

MySQL doesn’t provide scripts to monitor the replication process flow and alert operators when a problem occurs. For the LANL project, I wrote scripts to monitor the flow and server updates and to send alerts when the communications between master and slave servers failed.
Some replication management utilities have since become available, such as MySQL::Replication and My::Rep::MySQL Perl modules.

Optimization

Optimization can be a complicated task because it requires a deep understanding of the whole system. The most important component in speeding up a system is the basic design. It’s essential to know how the system behaves, how it will be used, and where the bottlenecks are.

Indexing

Indexing is the most important tool for speeding up queries. Indexes are used to quickly match rows. MySQL stores all indexes in B-trees. It can index all column types, though in the case of CHAR and VARCHAR, it’s much faster and requires less disk space to index a column prefix rather than the entire column. Using indexes on the relevant columns is the best way to make a query faster.

MySQL can also create n-column composite indexes—that is, indexes on multiple columns. A composite index serves as several indexes because MySQL can use any leftmost column set in the index to match rows. However, it cannot use the composite index for searches that don’t involve a leftmost prefix.

Overusing indexes can cause problems. Their performance benefits do come at a price. Every additional index takes disk space and lowers the performance of write operations. In addition, indexes must be organized and resorted when table contents change, especially with tables that contain variable-length columns. And the more indexes you have, the longer it takes.

Optimizing nontransactional tables is far less expensive with MySQL tools than SQL commands.

Disk issues

Disk access becomes important when dealing with databases of hundreds of gigabytes or more, where effective caching becomes impossible. Data distribution is very important, and disk seeks can become a big performance bottleneck.

You can move tables or databases from the MySQL database directory to other locations. The recommended way to do this is to link only databases to a different location. Nevertheless, as the database grows, it’s also useful to link some tables to distant locations, especially those that applications access concurrently. When using this approach

### Table 1

<table>
<thead>
<tr>
<th>Data storage</th>
<th>MySQL</th>
<th>PostgreSQL</th>
<th>Oracle</th>
<th>MSSQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage models</td>
<td>MyISAM, InnoDB, Berkeley DB, full-text</td>
<td>Postgres</td>
<td>Bitmapped, B-tree, IOT, function-based</td>
<td>Clustered, nonclustered</td>
</tr>
<tr>
<td>Reliability</td>
<td>High/very high</td>
<td>High</td>
<td>High/very high</td>
<td>High/very high</td>
</tr>
<tr>
<td>Scalability</td>
<td>Large/very large</td>
<td>Large</td>
<td>Large/very large</td>
<td>Large/very large</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indexes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single- and multicol.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>primary key, and full text</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data integrity</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID compliance, row-level locking, hot backup, and partial rollback</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Replication</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single master</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multimaster</td>
<td>Yes</td>
<td>Yes/no*</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Clustering</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface methods</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC/JDBC, C/C++, and Java</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Advanced features</th>
<th>Yes (starting with version 5.x)</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored procedures, views, triggers, sequences, and cursors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Solutions exist but they are commercial.*
you need to remember that, in MySQL versions prior to 4.0, some commands such as `ALTER`, `REPAIR`, and `OPTIMIZE TABLE` will remove the symbolic links and replace them with the original file. This happens because these statements create a temporary file in the database directory and replace the original file with the temporary file when the statement operation is complete.

**Query processing**

Complex queries can make it difficult to understand MySQL rules for deciding exactly how to fetch data. Fortunately, there are a few general rules and a command to help.

MySQL will not use an index if it decides that it would be faster to simply scan the entire table than, for example, to access roughly 30 percent of the table’s rows. If multiple indexes can satisfy a query, MySQL will use the most restrictive one—that is, the one that would fetch the fewest rows.

If the columns you’re selecting are all part of an index, MySQL might read all the data you need directly from the index and never access the table itself. When joining several tables, MySQL will first read data from the table that is likely to return the fewest rows. The order in which you specify the tables might not be the same order in which MySQL uses them. This also affects the order in which the rows are ultimately returned to you, so be sure to use an `ORDER BY` clause in your query if you need the rows in a particular order.

Having said all that, it’s important to realize that some MySQL decisions are actually based on guesses. You can, however, compel MySQL to use, ignore, or force a specific index or index set.

The `EXPLAIN` command will also help you understand what MySQL is doing to process a query.

**Query processing and LIMIT clause**

The `LIMIT` clause helps to control the number of rows returned, which is useful in the Web-search context when results must be displayed in chunks.

When you combine `LIMIT` with a query statement such as `SELECT`, MySQL handles the query differently depending on the number of rows requested and their location. If the selection involves only a few rows, MySQL uses indexes rather than doing a full table scan. With MySQL (as well as other RDBMSs) you can’t assume that results returned using `LIMIT` with subsequent chunks will return consecutive results without duplications. To avoid that problem, you must use `LIMIT` in conjunction with `ORDER BY` to sort the results. MySQL ends the sorting as soon as it finds the first lines requested with `LIMIT`.

In cases where the `ORDER BY` clause costs too much in performance terms, an alternative is to create a temporary table with the data and then apply the query with `LIMIT` on the temporary table.

**Query issues**

In general, after checking and eventually adding column indexes, you can speed up a slow query such as `SELECT ... WHERE` by reducing the `WHERE` clause as much as possible and moving the logic into the `SELECT` part whenever possible. MySQL evaluates the condition in the `WHERE` clause in a wider set, especially in a join context. The set returned on the `SELECT` is narrow and therefore easier and faster to manipulate than the `WHERE` clause.

Our experience on this project has been extremely positive. MySQL’s reputation for speed has proved itself in handling links among over 1.5 billions of rows of data in several virtual tables, laid over a terabyte of data. Moreover, we have used its replication capabilities to balance the load, reinforce fault tolerance, and protect and update our data. Our lessons learned should certainly help advocates promote MySQL and open source for building a sound technology infrastructure. Some of them might also interest practitioners or teams using other products.