Week 9
Introducing MySQL

Lesson 1: Introducing MySQL

In this lesson, you learn what the MySQL database system is and what it can do.

Database Basics

At its very simplest, a database is an organized way of holding together various pieces of information. The term database actually refers to the collection of data, not the means by which it is stored. For a computer database, the software used to manage the data is known as a database-management system (DBMS).

A database need not be stored on computer—although, of course, that is what this book is about! A filing cabinet and a telephone directory both contain databases: They hold information, organized in a structured manner so that you can easily access the individual item you want to find.

Relational Databases

MySQL is a relational database-management system (RDBMS). The term relational indicates that MySQL can store its data as a number of different tables that are related to each other in some way.

The advantage of this type of database over a flat table system is that very large databases can be constructed from different tables, each of which contains only information relevant to that table.

Much has been written about relational database theory, but you need to understand only a little to create efficient databases. You don't need to be scared by what appears to be a very academic topic—a great deal of the principles of relational database design are grounded in common sense.

Why Use MySQL?

You can choose from many different RDBMS options, so why use MySQL over another system?

One of the primary factors when choosing an RDBMS is cost. MySQL is distributed as open-source software under the GNU General Public License, so you can actually use MySQL free of charge.

Also available for MySQL is a commercial license that includes various levels of technical support for users with mission-critical systems. Even with the cost of commercial support, MySQL offers a significantly lower total cost of ownership over other enterprise-level RDBMS.

MySQL is robust, powerful, and scalable. It can be used in applications from tiny web databases to very large data warehouses with terabytes of data. You can use MySQL replication or clustering to guarantee 100% availability. A number of case studies published on MySQL.com give an example of just what MySQL is capable of.

Anatomy of a Database

A database consists of a series of tables. Each table has a name, which is how it is referenced in the SQL language. Each database also has a name, and the same RDBMS can manage many different databases. MySQL is a multiuser database and can restrict access to each database to only specific users.

A database table looks somewhat like a spreadsheet: a set of rows and columns that can contain data at each intersection. In a relational database, you store different types of data in different tables. For instance, the sample database in this book has separate tables for customers, products, and orders.

Each table has a defined set of columns, which determine the values it can store. For example, the table that contains information about products needs to store a name, price, and weight for every product. Each row of data can contain values for only the columns defined in the table.

In addition, each column is defined as a particular data type, which determines the kind of values it can store. The data type might restrict values to numeric or date values, or it might impose a maximum size.

What Is SQL?

The Structured Query Language (SQL) is the most common way to retrieve and manage database information. An SQL query consists of a series of keywords that define the data set you want to fetch from the database. SQL uses descriptive English keywords, so most queries are easy to understand.

SQL

Sometimes you will hear SQL pronounced as the word sequel, but it's also okay to pronounce it as the letters S-Q-L.

Virtually every RDBMS on the market uses the SQL language. The MySQL implementation of SQL conforms to the ANSI SQL standards and implements some of its own extension to handle features that are specific to MySQL.

About MySQL AB and mysql.com

MySQL AB is the Swedish company that develops, maintains, and markets the MySQL database server and tools.

MySQL Source

Although MySQL is distributed as open-source software, MySQL AB owns the source code to MySQL and determines the terms under which it is made available. Currently, MySQL is distributed under the GNU General Public License.

MySQL AB provides technical support to users with commercial support packages. The company also runs MySQL training courses around the world and issues MySQL Certification to those who want to become accredited MySQL Developers or Database Administrators.

The MySQL website, http://www.mysql.com/, contains a section called Developer Zone (which you can access directly at http://dev.MySQL.com/) where you can find online support in the form of forums, mailing lists, and user groups. Consider joining one of the online communities; they are a great way to learn more about MySQL from other users.

You can download or view online (in the Developer Zone) a comprehensive reference manual for MySQL. You can download a printable PDF version or browse an HTML-format manual that is fully searchable.

Online Manual: You can use the shortcut URL http://www.mysql.com/ANYTHING to search the online manual for ANYTHING. If there is an exact match, you will be taken straight to the appropriate page; otherwise, you will see a list of possible matches.

MySQL Components

Let's take a moment to look at where MySQL will reside on your system.

Linux/UNIX Systems

The location of MySQL programs, libraries, and other files depends on the installation prefix used when MySQL was installed. Typically, the prefix is either /usr/MySQL or /usr/local/MySQL.

In the location used on your system, you will find the following subdirectories:

- bin— Contains the MySQL executables, including the database server and all the client programs
- lib— Contains the development libraries used to communicate with a MySQL database from your own programs
- include— Contains the header files required to use MySQL APIs
- data— The MySQL data directory, containing the actual database files
- support_files— A number of sample configuration files

The MySQL configuration file is named my.cnf. The file /etc/my.cnf contains global settings, but you can also create a file named my.cnf in the data directory that applies only to that MySQL server; it is possible to have multiple MySQL servers on one machine.

RPMs

If you used a packaged installation method such as RPM, the system programs might have been installed to common locations alongside other applications. In this case, the data directory is /var/lib/MySQL.

Path

Make sure that the MySQL bin directory is in your system path so you can avoid having to enter the full path to the client programs when you want to run them. Refer to your operating system documentation for details on how to set this up.

Windows Systems

The default install location is C:\Program Files\MySQL. The system is installed in a subfolder that is named using the MySQL minor version number. For example, MySQL 5.0.18 is installed to C:\Program Files\MySQL\MySQL Server 5.0.

In the version-specific install location, you will find the following subdirectories:

- bin— The MySQL server and client program executables
- data— The MySQL data directory, containing the actual database files

In the installation directory, you will also find a number of sample configuration files. The current configuration is found in my.ini located in this folder.

MySQL is installed as a Windows service. To start and stop the database server, go to Control Panel, Administrativ Tools, and select Services. A program group in the Start menu contains a shortcut to the MySQL monitor program and the configuration wizard.

Lesson 2. Using MySQL

In this lesson, you learn how to connect to a MySQL database and use the mysql command-line program.

The mysql Client

The mysql program is a command-line client used for sending commands to a MySQL server. It can be used to enter SQL commands to query a database or alter table definitions; it also has its own set of commands to control its operation.

The mysql Program

In this lesson and throughout this book, mysql—printed in lower case—refers specifically to the MySQL command-line client program.

Starting the Command-Line Client

To start the command-line client, simply invoke the mysql program from the command line. If your local MySQL server allows anonymous connections, you can invoke mysql without any additional switches. The result looks similar to the following:

```
$ mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2084 to server version: 4.1.12-standard-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The mysql> prompt indicates that you will now be typing commands into the mysql client instead of the system shell.

When MySQL is first installed, anonymous connections are allowed unless you disable them. It is a good idea to remove anonymous user access. If your server does not allow anonymous connections, you will see an error like the following when you attempt to start the mysql program:

```
$ mysql
ERROR 1045 (28000): Access denied for user 'chris'@'localhost'
(using password: NO)
```

If you will be using mysql to connect to a remote database and there is no local MySQL server, or if the local MySQL server is not running, the error will look like this:

```
$ mysql
ERROR 2002 (HY000): Can't connect to local MySQL server through
socket '/var/lib/mysql/mysql.sock' (2)
```

Connecting to MySQL

Even if your database does allow anonymous connections, you will need to connect using a username and password to do anything useful. To connect to a MySQL database, you must know the connection parameters to use.

Because MySQL can accept connections over a network, you must specify the database server hostname. If you are connecting to a MySQL server running on the local machine, the hostname is localhost. Otherwise, you can use the IP address or hostname of the remote server.

You also need to supply a username and password to authenticate with the database server. The username and password to access a database can be locked down so that they work only when you connect from a specific location.

Finally, you must supply the database name to connect to. Each username may have access to one or more databases on the server; you must authenticate to gain access to your own databases.

The mysql program accepts a number of switches to specify how to connect to a database. Use the - -user switch to supply a username, and use the - -password switch to supply a password, as shown:

```
$ mysql --user=yourname --password=yourpass
```

Note that using this command as shown means that your password is visible onscreen. To avoid this, use the - -password switch without an argument to be presented with a password entry prompt.

$ mysql --user=yourname --password
Enter password:

Enter your password when prompted to log on to the MySQL server. Note that your password is not displayed onscreen as you type.

Windows Menu

On Windows systems, there is a menu item in the MySQL program group named MySQL Command Line Client. Selecting this item invokes mysql with a connection to the local server using the username root; you are prompted to enter a password to continue.

If there is a problem authenticating with the MySQL server using either of these methods, you will see an error like this:

ERROR 1045 (28000): Access denied for user 'yourname'@'localhost'
(using password: YES)

To connect to a database on a remote server, use the - -host switch.

$ mysql - -host=host.yourdomain.com - -user=yourname - -password

The value given in - -host can be the IP address of the server or a fully qualified domain name.

Remote Connections

Remember that your firewall must be configured to allow you to access the MySQL port if you need to connect to a remote database.

Each of the connection parameter switches to mysql has a shorter version that you can use, if you prefer. The - -user switch can be replaced by -u, and - -password by - p. When using the shorter switches, note that the equals sign is not needed—the value is given immediately after the switch.

For instance, the following two commands are identical:

$ mysql --user=yourname - -password
$ mysql -uyourname -p

The - - host switch can be abbreviated to - h in the same way, as shown in the following command:

$ mysql - hhost.yourdomain.com - uyourname - p

MySQL Port

The default port for connections to MySQL over a network is 3306. If your server uses a different port number, specify it using - -port= or -P when connecting using mysql.

The - -database switch can be abbreviated to -D. In fact, the - -database or -D switch can be omitted, and you can just specify the database at the end of the mysql command, as shown:

$ mysql-hhost.yourdomain.com -uyourname -ppassword dbname

Password Argument

When using the short connection switches, make sure that there is no space between -p and the password. Otherwise, you will be prompted to enter a password, and the password you gave on the command line will be treated as the database name.

Executing SQL Statements

Now let's look at how the mysql program is used to execute a SQL statement on a MySQL database.

Selecting a Database to Use

Your username and password give you access to one or more specific databases. For instance, if you are using a MySQL server provided by your web host, your username will give you access only to the database that is included with your web space. Other customers have their own individual passwords.

When you are connected to MySQL, issue the show databases command to see which databases are available to you. The SHOW DATABASES command is specific to MySQL but is executed through the mysql program like all other SQL commands.

mysql> Show Databases;
+--------------------+
| Database |

```
+--------------------+
| mysql              |
| test               |
| yourdb             |
+--------------------+
3 rows in set (0.00 sec)
```

To select the database named yourdb, use the \u command along with the database name.

```
mysql> \u yourdb
Database changed
```

Built-In Commands

Commands that begin with a backslash are internal commands for the mysql monitor program, not part of the SQL command set.

If you attempt to connect to a database that does not exist, or if you mistype the name, you will see an error message.

```
mysql> \u wrongdb
ERROR 1049 (42000): Unknown database 'wrongdb'
```

You can also specify the database to connect to by using the - -database or -D switch to the mysql program.

```
$ mysql - -user=yourname - -database=yourdb - -password
```

In fact, the mysql program enables you to put a database name after all the switches have been given without prefixing it with - -database or -D. The following is, therefore, equivalent to the previous command:

```
$ mysql - -user=yourname - -password yourdb
```

Showing Connection Status

The \s command returns the status of the current database connection, including the current database name and the connection parameters.

If you are using several databases at the same time, issuing \s is a handy way to find out which one you are working on at any given time. The following output shows a connection to a MySQL 4.1 server running on a local Windows machine:

```
mysql> \s
- - - - - - -
C:\Program Files\MySQL\MySQL Server 4.1\bin\mysql.exe
Ver 14.7 Distrib 4.1.14, for Win32 (ia32)
Connection id:         94
Current database:      mysqlin10
Current user:          root@localhost
SSL:                   Not in use
Using delimiter:       ;
Server version:        4.1.14-nt
Protocol version:      10
Connection:            localhost via TCP/IP
Server characterset:   latin1
Db     characterset:   latin1
Client characterset:   latin1
Conn.  characterset:   latin1
TCP port:              3306
Uptime:                35 days 5 hours 34 min 51 sec

Threads: 1  Questions:  542843 Slow queries: 18 Opens: 546
Flush tables: 1 Open tables: 0 Queries per second avg: 0.178
- - - - - - - -
```

Reconnecting

To force a reconnect to the database server, issue the \r command.

Entering a SQL Command

Although you will not learn specific SQL commands until the next lesson, here you should understand how to execute SQL statements through mysql.

Because SQL commands can span many lines, you must use a semicolon to tell MySQL that a command is

finished.

Look below at the SHOW DATABASES command, You'll see that it required a semicolon at the end of the command. If this had been omitted, the text on your screen would look like this instead:

```
mysql> SHOW DATABASES
   ->
```

When you press the Enter key and a command has not been terminated, the prompt changes from mysql> to ->, indicating that you are continuing a command from the previous line of input. You can enter a semicolon at any time to terminate the statement, and your SQL command can span as many lines as necessary before being terminated.

Consider the following SQL query (you do not need to understand how it works yet), entered in mysql. The result of the query from the sample tables used in the book appears directly underneath the query.

```
mysql> SELECT first_name, last_name
   -> FROM customer_contacts
   -> WHERE customer_code = 'PRESINC'
   -> ORDER BY last_name;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Abraham    | Lincoln   |
| Richard    | Nixon     |
| Franklin   | Roosevelt |
| Theodore   | Roosevelt |
+------------+-----------+
4 rows in set (0.00 sec)
```

The mysql program enables you to continue entering the query one line at a time; it sends the query to the MySQL server for execution only when you enter the semicolon to terminate the statement.

The query results are displayed in tabular format, just as you saw for SHOW DATABASES. The only difference this time is that there are two columns in the table. This format will quickly become familiar—it is the default output format for all SQL queries executed through mysql.

Underneath the query results is shown the total number of records retrieved by the query and the total execution time required to produce the results. In this example, the execution time appears to be zero—in fact, the time taken by the MySQL server was less than one hundredth of a second. Only when you begin to create complex queries will you notice a significant execution time.

When No Records Are Retrieved

If your query returns no rows, MySQL returns no data and no error message. The following example shows the response produced:

```
mysql> SELECT *
   -> FROM emptytable;
Empty set (0.00 sec)
```

Instead of the row count, mysql shows the message Empty set. The execution time is still displayed.

Query Output Formats

Instead of using a semicolon terminator symbol, the mysql client provides two built-in commands that send your query to the MySQL server. The \g command immediately executes the query being entered and displays the results onscreen.

However, if you use \G instead of \g, the query results are displayed in a vertical format instead of the usual tabular layout. This format produces one row of output for each column in the database, as shown:

```
mysql> SELECT * FROM products
   -> \G
*************************** 1. row ***************************
  code: MINI
  name: Small product
weight: 1.50
 price: 5.99
*************************** 2. row ***************************
  code: MIDI
  name: Medium product
weight: 4.50
 price: 9.99
*************************** 3. row ***************************
  code: MAXI
  name: Large product
weight: 8.00
 price: 15.99
3 rows in set (0.02 sec)
```

Vertical Format

The vertical query format is very useful when you want to view a single row of data from a table. The tabular layout might produce output that is too wide for your screen if the table contains many columns.

You can also change the default output format to vertical by starting mysql with the - -vertical switch. Then simply terminating a query with a semicolon will produce output in this format.

Other Formats

Other format options that can be selected with switches when mysql is invoked are - -html and - -xml. These produce HTML table and XML data output, respectively.

Editing a SQL Command

MySQL provides the capability to easily edit the last SQL statement you entered so that you can make a small change or correction to your query without needing to retype it in full.

Use the \e command to open an editor containing the current query. The actual editor used depends on your system environment; this is usually vi or vim, unless you specify otherwise.

Changing Editors

To change the query editor, set the name of the program you want to use in the EDITOR environment variable.

After you edit and save the query, you are returned to the mysql client with the prompt showing ->. It is as if you had just entered the query at the command line. To execute the query again, enter a semicolon or use the \g or \G commands.

Most modern systems also support command-line editing. You can use the left and right cursor keys to move along the line currently being entered to change text as you type. You can also use the up and down keys to cycle through the command history and retrieve commands that you entered earlier in the session.

Starting Over

If you want to start again with a new SQL statement without running the command being entered, issue the \c command to clear the current command and return to the mysql> prompt.

Capturing Output from MySQL

If you want to divert the output of a SQL query to a log file, use the \T command along with a filename. For instance, to write the output of a query to query.txt, do the following:

```
mysql> \T output.txt
Logging to file 'query.txt'
```

Any queries subsequently executed in mysql will be appended to query.txt and will be displayed onscreen. The retrieved data and the query itself are both written to the file—in fact, the log file records exactly what is displayed onscreen.

To stop writing to a log file, the \t command cancels any previously issued \T commands.

Exiting the mysql Program

To exit the mysql program, use the \q command. Alternatively, you can type quit or exit to leave the program.

If you want to run a single host command without exiting mysql, use the \! command. The following example runs the system command pwd on a UNIX/Linux system to determine the path of the current working directory:

```
mysql> \! pwd
/home/chris/public_html
```

Lesson 3: Retrieving Data

In this lesson, you learn how to use a SELECT statement to fetch records from a MySQL database.

The SELECT Statement

The first SQL command you will learn, and the one you will use most frequently, is SELECT. In this lesson, you begin by learning how to fetch data records from a single table.

A SELECT statement begins with the SELECT keyword and is used to retrieve information from MySQL database tables. You must specify the table name to fetch data from—using the FROM keyword—and one

or more columns that you want to retrieve from that table.

Keywords and Statements

A keyword is a word that is part of the SQL language. In the examples in this book, SQL keywords are always written in capitals, although they are not case sensitive.

A SQL statement begins with a keyword and can contain several more keywords that must appear in the correct, structured way—known as the statement's syntax.

Retrieving Individual Columns

If you execute the following SQL statement using mysql, the output produced will be as shown:

```
mysql> SELECT name
    -> FROM customers;
+-------------------------+
| name                    |
+-------------------------+
| Presidents Incorporated |
| Science Corporation     |
| Musicians of America    |
+-------------------------+
3 rows in set (0.02 sec)
```

Terminating a Statement

Remember that the semicolon character is required to indicate the end of a SQL statement. Alternatively, you can use the go command or \g to tell mysql to execute a query.

The customers table contains three records. In this statement, we tell MySQL to fetch the value of the name column; this is displayed for every record in the table.

The data displayed is not ordered. Usually records are retrieved in the same order in which they were inserted into the database. In this example, the company names are displayed in the order in which they were inserted in the sample table-creation script.

Order of Records

Although records are normally retrieved in the order in which they are inserted into the database, you cannot rely on a particular order being preserved. If your database is backed up and restored, or if a maintenance operation is performed on the database, MySQL might alter the order in which records are stored internally.

A SELECT statement will return every row from the table unless you tell it otherwise. You will learn how to do this, by putting a filter on the query, in the next lesson.

Retrieving Multiple Columns

Now you'll try another simple SELECT statement, this time on the products table. You can retrieve the values from two columns in the same query by specifying a list of columns after the SELECT keyword, separating them with a comma.

```
mysql> SELECT name, price
    -> FROM products;
+----------------+-------+
| name           | price |
+----------------+-------+
| Small product  |  5.99 |
| Medium product |  9.99 |
| Large product  | 15.99 |
+----------------+-------+
3 rows in set (0.01 sec)
```

The columns in the output appear in the order given in the query. To add the weight column to the data retrieved, add it to the end of the list of columns selected, as follows:

```
mysql> SELECT name, price, weight
    -> FROM products;
+----------------+-------+---------+
| name           | price |  weight |
+----------------+-------+---------+
| Small product  |  5.99 |    1.50 |
| Medium product |  9.99 |    4.50 |
| Large product  | 15.99 |    8.00 |
+----------------+-------+---------+
```

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

Formatting Queries

In the examples in this book, queries are formatted so that each clause of a SQL statement is on a separate line. The formatting does not affect the operation of a SQL statement; you can use carriage returns and whitespace to format a query however you want.

Although it might seem excessive to adopt a formatting style for these simple examples, as you add more clauses to a query and learn to build more complex SQL statements in subsequent lessons, formatting your queries in a readable way becomes very important.

Retrieving All Columns

If you want to retrieve the data from every column in a table, you do not need to specify each column name after the SELECT keyword. Use the asterisk character (*) in place of a column list in a SELECT statement to instruct MySQL to return every column from the specified table.

The following query retrieves every column and row from the products table:

```
mysql> SELECT *
   -> FROM products;
+------+----------------+--------+-------+
| code | name           | weight | price |
+------+----------------+--------+-------+
| MINI | Small product  |   1.50 |  5.99 |
| MIDI | Medium product |   4.50 |  9.99 |
| MAXI | Large product  |   8.00 | 15.99 |
+------+----------------+--------+-------+
3 rows in set (0.00 sec)
```

Note that the output produced is exactly the same, as if you had specified each column in the query by name, like this:

```
mysql> SELECT code, name, weight, price
   -> FROM products;
+------+----------------+--------+-------+
| code | name           | weight | price |
+------+----------------+--------+-------+
| MINI | Small product  |   1.50 |  5.99 |
| MIDI | Medium product |   4.50 |  9.99 |
| MAXI | Large product  |   8.00 | 15.99 |
+------+----------------+--------+-------+
3 rows in set (0.00 sec)
```

When you use SELECT *, columns are displayed in the order they occur in the database table—the order in which columns were specified when the table was created.

Compare the order of the columns in the result of the previous query to the output produced by the DESCRIBE command for products.

```
mysql> DESCRIBE products;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| code   | varchar(10)  |      |     |         |       |
| name   | varchar(40)  |      |     |         |       |
| weight | decimal(6,2) |      |     | 0.00    |       |
| price  | decimal(6,2) |      |     | 0.00    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Query Results

The tabular layout that mysql produces when displaying the results of a query is a simple yet readable format for viewing small amounts of data. However, your screen has only a fixed width, so if you try to display too many columns—or if a SELECT * returns a lot of columns—the characters that make up this table could wrap lines and create an unreadable mess.

Mistakes in a SELECT Statement

Before long, you will mistype a SELECT statement—if you have not done so already! Here you look at some of the error messages MySQLgives when you make a mistake.

If you try to select data from a table that does not exist, MySQL gives an error message. In this example, you attempted to select from a table named product instead of products:

```
mysql> SELECT *
   -> FROM product;
ERROR 1146 (42S02): Table sampdb.product' doesn't exist
```

If you specify a column name that does not exist in the selected table, you will see the following error message:

```
mysql> SELECT name
   -> FROM customer_contacts;
ERROR 1054 (42S22): Unknown column 'name' in 'field list'
```

In this case, the customer_contacts table does not have a name column—it has separate first_name and last_name columns.

Case Sensitivity

MySQL table names are case sensitive, but column names are not. A table named Products is different than products, and executing SELECT * FROM Products will produce an error in the sample database. However, specifying the name column as Name or NAME will not cause an error.

If you make a syntax error—that is, when MySQL cannot understand the SELECT statement because things do not appear in the order that it expects them to—the error message looks like the following:

```
mysql> SELECT first_name, last name,
   -> FROM customer_contacts;
ERROR 1064 (42000): You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for
the right syntax to use near 'FROM customer_contacts' at line 2
```

In the previous example, note the comma after last_name. When MySQL sees this, it expects another column name to follow, but instead the next word is FROM. Because you cannot use a SQL keyword as a column name, this causes the syntax error as shown.

MySQL can also throw up a syntax error if you misspell a keyword. In the following example, the keyword FROM was mistyped as FORM. The error displayed indicates that MySQL does not know what purpose the word FORM serves in the SQL statement, so it cannot execute this query.

```
mysql> SELECT name
   -> FORM products;
ERROR 1064 (42000): You have an error in your SQL syntax; check
the manual that corresponds to your MySQL server version for
the right syntax to use near 'products' at line 2
```

Multiple Errors

If there are several errors in your query, MySQL will tell you only about the first one that it encounters. Of the three error types shown previously, MySQL will report a syntax error first, followed by a nonexistent table and finally a bad column name.

Consider the error in the following statement, in which you mistyped a two-word column name by leaving out the underscore character in the name. MySQL does not allow column names to contain a space, so an underscore is often used to separate words.

```
mysql> SELECT_first name, last name
   -> FROM customer_contacts;
ERROR 1054 (42S22): Unknown column 'last' in 'field list'
```

In this example, MySQL gives an unknown column error instead of a syntax error. The actual way MySQL interprets this is to select a column named last and give it an alias name, so there's actually no error in the statement syntax.

Semantic Errors

The previous error is an example of a semantic error. The statement syntax is technically correct and is accepted by the MySQL interpreter, but the actual meaning of the statement is not what you intended it to be.

Retrieving Database Information

To construct a valid SELECT statement, you need to know how a database is organized. The SHOW command is used to retrieve information about database components.

Retrieving a List of Databases

Use the SHOW DATABASES command to retrieve a list of databases that you have access to. Execute the SHOW command just like a SELECT statement from the mysql program.

```
mysql> SHOW DATABASES;
+------------------+
| Database         |
+------------------+
| mysqlin10        |
| mydb             |
+------------------+
2 rows in set (0.00 sec)
```

Retrieving a List of Tables

When you have connected to a database with the use command in mysql, you can obtain a list of tables in that database with the SHOW TABLES command.

```
mysql> SHOW TABLES
+---------------------+
| Tables_in_mysqlin10 |
+---------------------+
| customer_contacts   |
| customers           |
| order_lines         |
| orders              |
| products            |
+---------------------+
5 rows in set (0.00 sec)
```

If you are connected to one database but want to list the tables in another, you can use a FROM clause with SHOW TABLES.

```
mysql> SHOW TABLES FROM sampdb;
+---------------------+
| Tables_in_sambdb    |
+---------------------+
| customer_contacts   |
| customers           |
| order_lines         |
| orders              |
| products            |
+---------------------+
5 rows in set (0.00 sec)
```

Retrieving a List of Columns

To retrieve the table structure for a database table, use the SHOW COLUMNS command using the table name in the FROM clause.

```
mysql> SHOW COLUMNS FROM products;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| code   | varchar(10)  |      |     |         |       |
| name   | varchar(40)  |      |     |         |       |
| weight | decimal(6,2) |      |     | 0.00    |       |
| price  | decimal(6,2) |      |     | 0.00    |       |
+--------+--------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Describe

The DESCRIBE command is a shortcut for the SHOW COLUMNS command; DESCRIBE table is identical to SHOW COLUMNS FROM table.

Lesson 4. Filtering and Sorting Data

In this lesson, you learn how to add a WHERE clause to a SELECT statement to filter the retrieved data, and how to use an ORDER BY clause to sort the query results.

The WHERE Clause

You can add a WHERE clause to a SELECT statement to tell MySQL to filter the query results based on a given rule. Rules in a WHERE clause refer to data values returned by the query, and only rows in which the values meet the criteria in the rule are returned.

Filtering on an Exact Value

The simplest type of filter in a WHERE clause uses the equals operator (=) to specify that a data value must match a given value exactly.

The following query retrieves all the contacts for a given company:

```
mysql> SELECT id, first_name, last_name
   -> FROM customer_contacts
   -> WHERE customer_code = 'SCICORP';
+----+------------+-----------+
| id | first_name | last_name |
+----+------------+-----------+
|  4 | Albert     | Einstein  |
|  5 | Charles    | Darwin    |
|  6 | Marie      | Curie     |
|  7 | Benjamin   | Franklin  |
+----+------------+-----------+
4 rows in set (0.03 sec)
```

In this example, rows from customer_contacts are returned only if the value of customer_code in that row is equal to SCICORP.

You often use an exact match operator in a WHERE clause to return all the columns in a table when you know the primary key of a record. For instance, to find all the information about a product from its unique product code, you could use the following query:

```
mysql> SELECT *
   -> FROM products
   -> WHERE code = 'MIDI';
+------+----------------+--------+-------+
| code | name           | weight | price |
+------+----------------+--------+-------+
| MIDI | Medium product |   4.50 |  9.99 |
+------+----------------+--------+-------+
1 row in set (0.00 sec)
```

To perform a condition using an inequality, use the(=)operator. This works just the same as =, but the condition returns only rows in which the table value is not equal to the value given in the condition.

For instance, the following query finds all products except for MINI:

```
mysql> SELECT *
   -> FROM products
   -> WHERE code(=)'MINI';
+------+----------------+--------+-------+
| code | name           | weight | price |
+------+----------------+--------+-------+
| MIDI | Medium product |   4.50 |  9.99 |
| MAXI | Large product  |   8.00 | 15.99 |
+------+----------------+--------+-------+
2 rows in set (0.02 sec)s
```

Filtering on a Range of Values

In addition to the equality operator, you can use a set of operators to select rows based on a range of values. Use the symbols < and > to test whether one value is less than or greater than another value, respectively.

When followed by an = symbol, these operators also match equal values. The symbol sequence >= means "is greater than or equal to," whereas <= means "is less than or equal to."

To find only products for which the price is $9.99 or lower, use the following query:

```
mysql> SELECT *
   -> FROM products
   -> WHERE price <= 9.99;
+------+----------------+--------+-------+
| code | name           | weight | price |
+------+----------------+--------+-------+
| MINI | Small product  |   1.50 |  5.99 |
| MIDI | Medium product |   4.50 |  9.99 |
+------+----------------+--------+-------+
2 rows in set (0.00 sec)
```

This example uses the <= operator, meaning that rows in which the price value is less than or equal to 9.99 are returned. If you use the < operator instead, the MIDI product will not be returned by the query because it costs exactly $9.99.

The range operators can be performed on textual data, and the results are logical. For example, performing a greater-than comparison on the last_name field returns only values that are alphabetically higher than the given value.

```
mysql> SELECT last_name
   -> FROM customer_contacts
   -> WHERE last_name > 'G';
+-----------+
| last_name |
+-----------+
| Lincoln   |
| Nixon     |
| Roosevelt |
| Gershwin  |
| Lennon    |
+-----------+
5 rows in set (0.00 sec)
```

Word Comparisons

Comparing text strings in MySQL works like the ordering of words in a dictionary. The word sandwich would appear after the word sand in the dictionary because it is a longer word. Similarly MySQL would consider sandwich to be a greater value than sand.

Using Quotes Around Values

You might have noticed in the previous examples that sometimes the values used in a WHERE clause were contained in single quotes. This is necessary when the value being compared is a non-numeric value because MySQL needs to know whether you are referring to a fixed value or a column name from a table in the query.

Consider the following query, which produces the error shown:

```
mysql> SELECT last_name
   -> FROM customer_contacts
   -> WHERE first_name = Benjamin;
ERROR 1054 (42S22): Unknown column 'Benjamin' in 'where clause'
```

The error message indicates that MySQL is trying to find a column named Benjamin in the customer_contacts table—of course, this does not exist. To tell MySQL that a value is not a column name, you must enclose it in quotes, as shown:

```
mysql> SELECT last_name
   -> FROM customer_contacts
   -> WHERE first_name = 'Benjamin';
+-----------+
| last_name |
+-----------+
| Franklin  |
| Britten   |
+-----------+
2 rows in set (0.00 sec)
```

Although the usual convention in a WHERE clause is WHERE column = ' value', this ordering is not significant. The following query demonstrates that the order can be reversed without affecting the outcome:

```
mysql> SELECT last_name
   -> FROM customer_contacts
   -> WHERE 'Benjamin' = first_name;
+-----------+
| last_name |
+-----------+
| Franklin  |
| Britten   |
+-----------+
2 rows in set (0.00 sec)
```

Number Values

Quotes are not needed around number values. MySQL does not get confused with numeric comparisons because a column name cannot begin with a number.

MySQL does enable you to perform a query using two columns, although this is rarely useful with two columns from the same table. A slightly peculiar example is shown here:

```
mysql> SELECT first_name, last_name
   -> FROM customer_contacts
```

```
   -> WHERE first_name = last_name;
Empty set (0.00 sec)
```

This query attempts to find names from the customer_contacts table whose first name and last name are the same. Such names are unusual—the query will find only names such as Scott Scott or Thomas Thomas—so no rows are returned from the sample tables. Note that this query does not find matches for Benjamin Franklin and Franklin Roosevelt—it compares only the first_name and last_name values from a single row of data.

Comparing Column Values

Having two different column names on either side of the = character in a WHERE clause is an important SQL technique for queries that use more than one table.

The ORDER BY Clause

So far, you have seen that data is fetched from the database in no particular order. To specify the sorting on the result of a query, you can add an ORDER BY clause.

Sorting on a Single Column

The following example retrieves all the products in order of price. The keywords ORDER BY are followed by the name of the column on which you want to sort.

```
mysql> SELECT *
   -> FROM products
   -> ORDER BY price;
+------+----------------+--------+-------+
| code | name           | weight | price |
+------+----------------+--------+-------+
| MINI | Small product  |   1.50 |  5.99 |
| MIDI | Medium product |   4.50 |  9.99 |
| MAXI | Large product  |   8.00 | 15.99 |
+------+----------------+--------+-------+
3 rows in set (0.00 sec)
```

Sort Columns

The sort column specified in ORDER BY does not actually have to appear in the list of columns after the SELECT keyword. You can, therefore, specify a sort order using a column that is not retrieved by the query.

When you want to add sorting to a query that is also filtered, the ORDER BY clause must appear after the WHERE clause. This example finds all the contacts for one customer sorted on the last name.

```
mysql> SELECT first_name, last_name
   -> FROM customer_contacts
   -> WHERE customer_code = 'SCICORP'
   -> ORDER BY last_name;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Marie      | Curie     |
| Charles    | Darwin    |
| Albert     | Einstein  |
| Benjamin   | Franklin  |
+------------+-----------+
4 rows in set (0.00 sec)
```

Sorting on Multiple Columns

The ORDER BY clause is formed in a similar way to the first line of a SELECT statement. If you want to specify a sort order that involves more than one column in the query, separate the column names with a comma.

The following query fetches data from the orders table in date order. The second sort column, customer_code, is used to specify the sorting when the values of order_date are the same.

```
mysql> SELECT order_date, customer_code
   -> FROM orders
   -> ORDER BY order_date, customer_code;
+------------+---------------+
| order_date | customer_code |
+------------+---------------+
| 2006-01-23 | PRESINC       |
| 2006-01-23 | SCICORP       |
| 2006-01-26 | PRESINC       |
```

```
| 2006-02-01 | MUSGRP       |
| 2006-02-02 | MUSGRP       |
| 2006-02-02 | SCICORP      |
| 2006-02-05 | SCICORP      |
+------------+---------------+
7 rows in set (0.00 sec)
```

Specifying Sort Order

By default, the ordering on a column specified in the ORDER BY clause is done in ascending order, either numerically or alphabetically, depending on the data type of the column.

To specify a descending sort direction, use the DESC keyword. The following example sorts the data from the products table with the heaviest at the top of the list.

```
mysql> SELECT *
   -> FROM products
   -> ORDER BY weight DESC;
+------+----------------+--------+-------+
| code | name           | weight | price |
+------+----------------+--------+-------+
| MAXI | Large product  |   8.00 | 15.99 |
| MIDI | Medium product |   4.50 |  9.99 |
| MINI | Small product  |   1.50 |  5.99 |
+------+----------------+--------+-------+
3 rows in set (0.00 sec)
```

Ascending Order

To explicitly specify ascending sort order, use the ASC keyword in place of DESC. The ASC keyword is optional, but it can be useful to indicate the sort order clearly in a query.

Multiple Sort Orders

When you specify more than one sort column, the keywords ASC and DESC can be used after each column name. You must specify the order direction for each sort in turn.

Lesson 5. Advanced Data Filtering

In this lesson, you learn about more conditional operators that can be used in a WHERE clause. You also learn how to combine conditions to perform more advanced filtering on a query.

Combining WHERE Clauses

The examples you have seen so far perform filtering on a query based on only a single condition. To provide greater control over a result set, MySQL enables you to combine a number of conditions in a WHERE clause. They are joined using the logical operator keywords AND and OR.

Using the AND Operator

After adding a WHERE clause to filter a query, you can filter the results further by adding another condition with the AND operator.

This is commonly used to restrict the query results based on the values of two or more columns, as shown in the following example:

```
mysql> SELECT * From orders
   -> WHERE customer_code = 'Scicorp'
   -> AND order_date >= '2006-02-01';
+----+---------------+------------+
| id | customer_code | order_date |
+----+---------------+------------+
|  4 | Scicorp       | 2006-02-02 |
|  5 | SCICORP       | 2006-02-05 |
+----+---------------+------------+
2 rows in set (0.00 sec)
```

This query returns only rows from the orders table in which both conditions hold true. The customer_code must be SCICORP, and the order_date must be on or after February 1, 2006.

AND

For a query row to be returned, the table data in question must satisfy all the conditions separated by an And operator. If any one condition fails for a record, that record is filtered out.

Using the OR Operator

Whereas the AND operator specifies a filter that further restricts the number of rows returned by a query, the OR operator is used to relax the filtering criteria by specifying alternative filter conditions. If one or more of the conditions separated by an OR operator hold true for a row in the table, that record will appear in the query results.

The following example shows the same example you saw for the AND operator, but using an OR instead:

```
mysql> SELECT * FROM orders
   -> WHERE customer_code = 'Scicorp'
   -> OR order_date >= '2006-02-01';
+----+---------------+------------+
| id | customer_code | order_date |
+----+---------------+------------+
|  3 | SCICORP       | 2006-01-23 |
|  4 | SCICORP       | 2006-02-02 |
|  5 | SCICORP       | 2006-02-05 |
|  6 | MUSGRP        | 2006-02-01 |
|  7 | MUSGRP        | 2006-02-02 |
+----+---------------+------------+
5 rows in set (0.00 sec)
```

The result set for this query includes all orders for SCICORP, regardless of order_date, as well as any orders for other customers that were placed on or after February 1.

OR

For a query row to be returned, the table data in question must satisfy at least one of the conditions separated by an OR operator. Every condition must fail for a record to be filtered out.

This example finds all the customers who have the name Franklin, whether that is their first name or last name:

```
mysql> SELECT first_name, last_name
   -> FROM customer_contacts
   -> WHERE first_name = 'Franklin'
   -> OR last_name = 'Franklin';
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Franklin   | Roosevelt |
| Benjamin   | Franklin  |
+------------+-----------+
2 rows in set (0.00 sec)
```

Filtering on Multiple Values

You could use the OR operator with equals conditions to specify multiple filter values for the same column. For instance, the following query finds all the customer contacts called either Benjamin or Charles:

```
mysql> SELECT first_name, last_name
   -> FROM customer_contacts
   -> WHERE first_name = 'Benjamin'
   -> OR first_name = 'Charles';
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Charles    | Darwin    |
| Benjamin   | Franklin  |
| Benjamin   | Britten   |
+------------+-----------+
3 rows in set (0.00 sec)
```

However, because this is a relatively common type of filter, there is a handy shortcut. You can use the IN operator to perform exactly this type of filter in a single condition. IN works like multiple equals operators and takes a comma-separated list of values, enclosed in parentheses.

Brackets

The proper name for the brackets used in MySQL is parentheses. Because MySQL uses only parentheses, however, you can refer to them as brackets without confusing them with square brackets or braces.

The following query is equivalent to the previous example:

```
mysql> SELECT first_name, last_name
   -> FROM customer_contacts
   -> WHERE first_name In ('Benjamin', 'Charles');
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Charles    | Darwin    |
| Benjamin   | Franklin  |
| Benjamin   | Britten   |
+------------+-----------+
3 rows in set (0.00 sec)
```

You can already see that this query is more concise and easier to read when you use IN.Imagine how cumbersome a query would become without it if you had to use the OR operator with a long list of values to compare.

Instead of matching against a list of values, the BETWEEN operator enables you to search for data within a given range. It takes two target values separated by the keyword AND. Rows in which the column data is between—and including—those values are returned. The following example finds only orders placedin a given week in January:

```
mysql> SELECT *
   -> FROM orders
   -> WHERE order_date Between '2006-01-24' AND '2006-01-30';
+----+---------------+------------+
| id | customer_code | order_date |
+----+---------------+------------+
|  2 | PRESINC       | 2006-01-26 |
|  3 | SCICORP       | 2006-01-23 |
+----+---------------+------------+
2 rows in set (0.00 sec)
```

Date Comparisons

The previous example used the BETWEEN operator to compare two dates. In MySQL, dates are compared as numeric values, with earlier dates having lower values than later dates.

Negating a Condition

Using the NOT keyword negates a condition—it makes it behave in the opposite way.

A powerful use of NOT is the NOT IN condition. Instead of specifying a list of values that a column value should match, you specify a list of values that should be excluded from the query result. The following query finds customers from a given company code but also excludes a list of names from the result:

```
mysql> SELECT first_name, last_name FROM customer_contacts
   -> WHERE customer_code = 'PRESINC'
   -> AND first_name NOT IN ('Abraham', 'Theodore');
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Richard    | Nixon     |
| Franklin   | Roosevelt |
+------------+-----------+
2 rows in set (0.00 sec)
```

Similarly, you can use the NOT operator with a BETWEEN statement to exclude a range of values from the query result. The following example finds only orders that were placed outside the period given:

```
mysql> SELECT * FROM orders
   -> WHERE order_date NOT BETWEEN '2006-01-26' AND '2006-02-03';
+----+---------------+------------+
| id | customer_code | order_date |
+----+---------------+------------+
|  1 | PRESINC       | 2006-01-23 |
|  3 | SCICORP       | 2006-01-23 |
|  5 | SCICORP       | 2006-02-05 |
+----+---------------+------------+
3 rows in set (0.00 sec)
```

Operator Precedence

Consider the following query, which contains both an AND and an OR operator:

```
mysql> SELECT customer_code, first_name, last_name
   -> FROM customer_contacts
```

```
   -> WHERE customer_code = 'SCICORP'
   -> AND first_name = 'Albert'
   -> OR first_name = 'Benjamin';
```

Before looking at the output from this query, try to determine what MySQL is actually looking for here. This query could filter the customer contact data in two possible ways.

You might expect that the query would return contacts for the company SCICORP with one of the first names given. Or you might think that it will return anyone named Albert from SCICORP, as well as any other contact named Benjamin. However, unless you know whether MySQL assigns more importance to an AND or an OR, you will not be able to say for sure without running the query.

The actual results from the previous query are shown here:

```
+---------------+------------+-----------+
| customer_code | first_name | last_name |
+---------------+------------+-----------+
| SCICORP       | Albert     | Einstein  |
| SCICORP       | Benjamin   | Franklin  |
| MUSGRP        | Benjamin   | Britten   |
+---------------+------------+-----------+
3 rows in set (0.00 sec)
```

As you can see, a Benjamin from a company other than SCICORP appears in the results. This is because MySQL treats an AND with higher precedence than an OR.The effect is that the conditions on either side of the AND are evaluated first, before the OR operator is considered.

Precedence

The order in which components of SQL statements are evaluated is called precedence. For conditional operators, AND has a higher precedence than OR, which, in turn, has a higher precedence than NOT.

To override the natural order of evaluation, you can put parentheses around conditions to indicate that they should be evaluated together. The following query uses parentheses to override the operator precedence of the previous example:

```
mysql> SELECT customer_code, first_name, last_name
   -> FROM customer_contacts
   -> WHERE customer_code = 'SCICORP'
   -> AND (first_name = 'Albert'
   -> OR first_name = 'Benjamin');
+---------------+------------+-----------+
| customer_code | first_name | last_name |
+---------------+------------+-----------+
| SCICORP       | Albert     | Einstein  |
| SCICORP       | Benjamin   | Franklin  |
+---------------+------------+-----------+
2 rows in set (0.00 sec)
```

This time, the two queries on either side of the OR operator are evaluated first, before the AND operator is considered. This query returns a row only if the customer_code condition is true and at least one of the conditions in the OR is met.

Using Parentheses

You can use parentheses to indicate the order of evaluation even if doing so will not affect thenatural operator precedence. This will make yourqueries more readable.

Limiting the Number of Rows Returned

If you are expecting a query to still return more rows that you want, even with the filtering from a WHERE clause applied, you can add a LIMIT clause to specify the maximum number of records to be returned.

Using a LIMIT Clause

The following example retrieves all the rows from the customer_contacts table, but the LIMIT clause restricts the number of rows returned to three:

```
mysql> SELECT first_name, last_name
   -> FROM customer_contacts
   -> ORDER BY last_name
   -> LIMIT 3;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| Benjamin   | Britten   |
| Marie      | Curie     |
| Charles    | Darwin    |
```

```
+------------+-----------+
3 rows in set (0.00 sec)
```

Limiting Sorted Records

The sorting specified in an ORDER BY clause is performed as if the full result of the query was fetched, regardless of the number of rows specified by the LIMIT clause.

By specifying sorting on a query column and using a LIMIT clause to limit the number of rows returned to just one, you can tell MySQL to return only the record with the highest or lowest value in that column. The following example returns only the most expensive product from the productstable, by sorting pricein descending order:

```
mysql> SELECT *
   -> FROM products
   -> ORDER BY price DESC
   -> LIMIT 1;
+------+---------------+--------+-------+
| code | name          | weight | price |
+------+---------------+--------+-------+
| MAXI | Large product |   8.00 | 15.99 |
+------+---------------+--------+-------+
1 row in set (0.00 sec)
```

Skipping Rows

If the LIMIT clause contains two numbers separated by a comma, the first is an offset argument and the second is the number of rows to return. The offset specifies the number of rows to skip before returning thefirst record.

The next two queries show this in action. First select all the customers' email addresses in alphabetical order and show just the first three rows.

```
mysql> SELECT email
   -> FROM customer_contacts
   -> ORDER BY email
   -> LIMIT 3;
+--------------------------+
| email                    |
+--------------------------+
| britten@musgrp.com       |
| curie@sciencecorp.com    |
| darwin@sciencecorp.com   |
+--------------------------+
3 rows in set (0.00 sec)
```

The next query uses an offset value to show the next three rows from the query result.

```
mysql> SELECT email
   -> FROM customer_contacts
   -> ORDER BY email
   -> LIMIT 3,3;
+--------------------------+
| email                    |
+--------------------------+
| einstein@sciencecorp.com |
| fdr@presidentsinc.com    |
| franklin@sciencecorp.com |
+--------------------------+
3 rows in set (0.00 sec)
```

Notice that there is no overlap here—the offset value of 3 causes the fourth, fifth, and sixth rows of the query result to be displayed.

Skipping Rows

Remember that the offset value in a LIMIT clause is the number of rows to skip, not the number of the first row to display. LIMIT 5 is equivalent to LIMIT 0,5—not LIMIT 1,5.

Creating and Modifying Tables

In this lesson, you learn how to use the SQL commands for managing tables and databases in MySQL. This subset of SQL is known as the Data Definition Language.

Before we look at how tables are managed in MySQL, we should examine the command set that is used to

create databases on a MySQL server.

Creating a New Database

You use the CREATE DATABASE command to create a new database. To create a new database named newdb, issue the following command. The output will be as shown if the database is created successfully.

```
mysql> CREATE DATABASE newdb;
Query OK, 1 row affected (0.02 sec)
```

Creating Database

Usually you execute this command when you are connected as the root user, but you can create databases when logged in as any user who has the CREATE privilege.

The database name can be up to 64 characters in length and can contain any character that is allowed in a directory name on your underlying operating system, except for the / and \ characters. The database name also cannot be one of the SQL reserved words.