

12

Intro to Databases

The Internet wouldn't be where it is today if not for the existence of databases. In fact, PHP probably wouldn't be as popular or as useful if not for its built-in support for numerous types of databases. This chapter will use MySQL as the example database management system (DBMS). Although MySQL—which is available for most platforms—may not be as powerful as the highest-end commercial database servers, MySQL has enough speed and functionality for most purposes. And its price—free for most uses—makes it the common choice for Web development.

This chapter walks through the development of a simple database for running a basic blog. Although you'll learn enough here to get started working with database, you will want to visit Appendix B, "Resources and Next Steps," once you've finished this chapter to find some references where you can learn more about the topic.

In This Chapter

Introduction to SQL	334
Connecting to MySQL	336
MySQL Error Handling	340
Creating and Selecting a Database	343
Creating a Table	347
Inserting Data into a Database	352
Securing Query Data	358
Retrieving Data from a Database	361
Deleting Data in a Database	366
Updating Data in a Database	372
Review and Pursue	378

Everywhere you see `mysql` type `mysql`

Introduction to SQL

A *database* is a collection of tables (made up of columns and rows) that stores information. Most databases are created, updated, and read using SQL (Structured Query Language). There are surprisingly few commands in SQL (**Table 12.1** lists the seven most important), which is both a blessing and a curse.

SQL was designed to be written a lot like the English language, which makes it very user friendly. But SQL is still extremely capable, even if it takes some thought to create more elaborate SQL statements with only the handful of available terms. In this chapter you'll learn how to execute all the fundamental SQL commands.

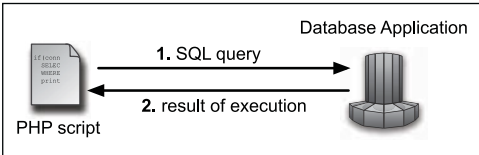
For people new to PHP, confusion can stem from PHP's relationship to HTML (i.e., PHP can be used to generate HTML but PHP code is never executed in the Web browser). When you incorporate a database, the relationships can become even fuzzier. The process is quite simple: PHP is used to send SQL statements to the database application, where they are executed. The result of the execution—the creation of a table, the insertion of a record, the retrieval of some records, or even an error—is then returned by the database to the PHP script **A**.

With that in mind, PHP's `mysql_query()` function will be the most-used tool in this chapter. It sends an SQL command to MySQL:

```
$result = mysql_query(SQL command,  
→ database connection);
```

TABLE 12.1 Common SQL Commands

Command	Purpose
ALTER	Modifies an existing table
CREATE	Creates a database or table
DELETE	Deletes records from a table
DROP	Deletes a database or table
INSERT	Adds records to a table
SELECT	Retrieves records from a table
UPDATE	Updates records in a table



A PHP will be used to send an SQL statement to MySQL. MySQL will execute the statement and return the result to the PHP script.

MySQL Support in PHP

Support for the MySQL database server has to be built into PHP in order for you to use PHP's MySQL-specific functions. For most PHP installations, this should already be the case. You can confirm support for MySQL by calling the `phpinfo()` function, which reveals details of your installation.

When working through this chapter, if you see an error message saying ... *undefined function mysql_...*, this means the version of PHP you're using doesn't have support for MySQL (or you misspelled the function name, which you should also check).

Enabling support for MySQL takes a little effort, but it can be done if you have administrative-level control over your server. For more information, see the PHP manual.

Everywhere you see mysql type mysql

I start this chapter with this prologue because the addition of SQL and MySQL to the Web development process will complicate things. When problems occur—and undoubtedly they will—you'll need to know how solve them.

When a PHP script that interacts with a MySQL database does not perform as expected, the first step is to determine if the problem is in the query itself—number 1 in **A**—or in the results of the query—number 2 in **A**. To take this step, you can start by printing out the query being executed, using code such as the following:

```
print $query;
```

Assuming that `$query` represents the complete SQL command, often containing the values of PHP variables, this one, simple line will reveal to you the actual SQL statement being run.

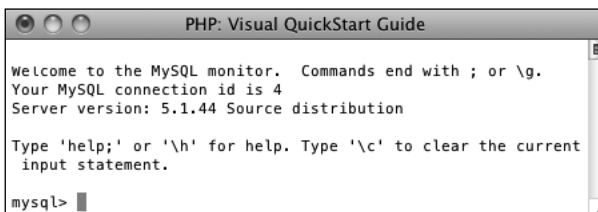
Next, you would take the printed query and execute it using another application. The two most common options are:

- The MySQL client **B**, a command-line tool for interacting with MySQL
- **phpMyAdmin C**, a PHP-based MySQL interface

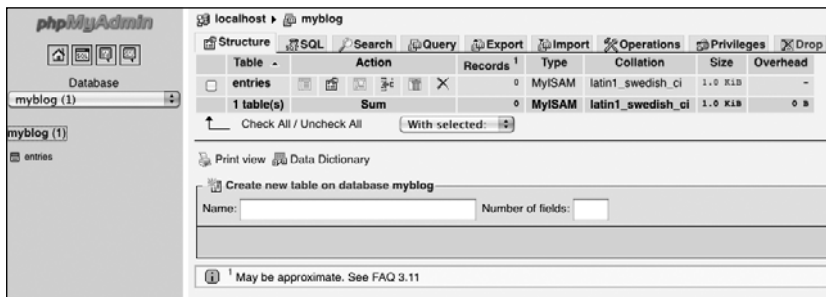
One or both of these should be provided to you by your hosting company or the software you installed on your own computer. For a demonstration of using each, see Appendix A, “Installation and Configuration.”

TIP Technically, a **DBMS**, or database application, is the software that interfaces with the database proper. However, most people use the terms *database* and *DBMS* synonymously.

TIP Lots of other applications are available for interacting with MySQL aside from the MySQL client and phpMyAdmin. Some are free, and others cost. A quick search using Google for *MySQL*, *admin*, and your operating system should turn up some interesting results.



B The MySQL client comes with the MySQL database software and can be used to execute queries without the need for a PHP script.



C phpMyAdmin is perhaps the most popular software written in PHP. It provides a Web-based interface for a MySQL database.

Connecting to MySQL

When you worked with text files in Chapter 11, “Files and Directories,” you saw that some functions, such as **fwrite()** and **fgets()**, require that you first create a file pointer using **fopen()**. This pointer then acts as a reference to that open file. You use a similar process when working with databases. First, you have to establish a connection to the database server (in this case, MySQL). This connection is then used as the access point for any future commands. The syntax for connecting to a database is

```
$dbc = mysql_connect(hostname,  
username, password);
```

The database connection (assigned to **\$dbc** in the above) is established using at least three arguments: the host, which is almost always *localhost*; the username; and the password for that username.

If you’re using a database through a hosting company, the company will most likely provide you with the host name, username, and password to use. If you’re running MySQL on your own computer, see Appendix A to learn how you create a user.

Once you’re done working with a database, you can close the connection, just as you’d close an open file:

```
mysql_close($dbc);
```

The PHP script will automatically close the database connection when the script terminates, but it’s considered good form to formally close the connection once it’s no longer needed.

For the first example of this chapter, you’ll write a simple script that attempts to connect to MySQL. Once you have this connection working, you can proceed through the rest of the chapter.

There are two things that needs to be done before using data from a database:

- Connect to the MySQL server
PHP function to connect to MySQL Server

The `mysqli_connect()` function takes four parameters in this exact order.

```
$dbc = mysqli_connect  
('localhost','root','password',  
'myblog');
```

- Connect to a database in the MySQL server
`mysqli_select_db (“myblog”);`

We are going to be using **mysqli**.

The **MySQLi Extension** ([MySQL Improved](#)) is a [relational database driver](#) used in the [PHP programming language](#) to provide an interface with [MySQL databases](#).

<http://en.wikipedia.org/wiki/MySQLi>

Everywhere you see mysql type mysqli

Script 12.1 Being able to connect to the MySQL server is the most important step. This script tests that process.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
  1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/
  xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/
  xhtml" xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="content-type"
     content="text/html; charset=utf-8" />
6   <title>Connect to MySQL</title>
7 </head>
8 <body>
9 <?php // Script 12.1 - mysql_connect.php
10 /* This script connects to the MySQL
   server. */
11
12 // Attempt to connect to MySQL and print
   out messages:
13 if ($dbc = mysql_connect('localhost',
   'username', 'password')) {
14
15   print '<p>Successfully connected to
     MySQL!</p>';
16
17   mysql_close($dbc); // Close the
     connection.
18
19 } else {
20
21   print '<p style="color: red;">Could
     not connect to MySQL.</p>';
22
23 }
24
25 ?>
26 </body>
27 </html>
```

`$dbc = mysqli_connect
('localhost','root','password',
'myblogXX');`

To connect to MySQL:

1. Begin a new PHP document in your text editor or IDE, to be named `mysql_connect.php` (Script 12.1):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/
   → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
   <meta http-equiv="content-type"
   → content="text/html;
   → charset=utf-8" />
   <title>Connect to MySQL</title>
</head>
<body>
```

2. Start the section of PHP code:

```
<?php // Script 12.1 -
→ mysql_connect.php
```

3. Connect to MySQL, and report on the results:

```
if ($dbc = mysql_connect
→ ('localhost', 'username',
→ 'password')) {
   print '<p>Successfully connected
   → to MySQL!</p>';
   mysql_close($dbc);
} else {
   print '<p style="color:
   → red;">Could not connect to
   → MySQL.</p>';
}
```

By placing the connection attempt as the condition in an `if-else` statement, you make it easy to report on whether the connection worked.

This chapter will continue to use `username` and `password` as values.

continues on next page

Everywhere you see mysql type mysqli

For your scripts, you'll need to replace these with the values provided by your Web host or set them when you add a user using the steps outlined in Appendix A.

If a connection was established, a positive message is printed and then the connection is closed. Otherwise, a message stating the opposite is printed, and there is no need to close the database connection (because it wasn't opened).

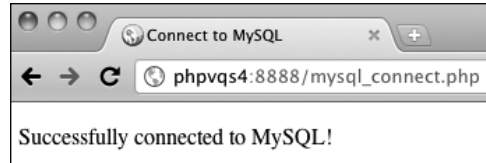
4. Complete the PHP code and the HTML page:

```
?>
</body>
</html>
```

5. Save the file as **mysql_connect.php**, place it in the proper directory of your PHP-enabled computer, and test it in your Web browser **A**.

If you see results like those in **B**, double-check the username and password values. They should match up with those provided to you by your Web host or those you used to create the user. You can always test your connection username and password by using them in the MySQL client (again, see Appendix A).

If you see *call to undefined function mysql_connect...*, your version of PHP doesn't support MySQL (see the "MySQL Support in PHP" sidebar).



A If PHP has support for MySQL and the username/password/host combination you used was correct, you should see this simple message.



B If PHP couldn't connect to MySQL, you'll probably see something like this. The warning message may or may not appear, depending on your error management settings.

MySQL Extensions

PHP can communicate with MySQL using different extensions. The first, used in this chapter, is the “standard” MySQL extension. It has been around for years and works with all versions of PHP and MySQL. All of the standard MySQL extension functions begin with *mysql_*.

The second extension is called *MySQLi* (Improved MySQL Extension). This extension was added in PHP 5 and can be used with MySQL 4.1 or greater. These functions all begin with *mysqli_* and take advantage of some of the added features in MySQL. If possible, it's preferable to use the *MySQLi* functions, but as the older extension is more universally enabled, this book uses it exclusively. See the PHP manual or my book *PHP 6 and MySQL 5 for Dynamic Web Sites: Visual QuickPro Guide* (Peachpit Press, 2007) for details on the *MySQLi* extension.

TIP The database connection (`$dbc` in this case) in most `mysql_something()` functions is optional. Regardless of that, in this book you'll see it always used, as it's not optional in the `mysqli_something()` functions and you should be prepared to provide the database connection when you move to those functions later (see the “MySQL Extensions” sidebar).

TIP The `localhost` value is used as the hostname when both the PHP script and the MySQL database reside on the same computer. You can use PHP to connect to a MySQL database running on a remote server by changing the hostname in the PHP script and creating the proper permissions in MySQL.

TIP PHP has built-in support for most databases, including dBase, FilePro, SQLite, MySQL, Oracle, PostgreSQL, and Sybase. If you're using a type of database that doesn't have direct support—for example, Access or SQL Server—you'll need to use PHP's ODBC (Open Database Connectivity) functions along with that database's ODBC drivers to interface with the database.

TIP The combination of using PHP and MySQL is so common that you may run across terms that identify servers configured with both PHP and MySQL: *LAMP*, *MAMP*, and *WAMP*. These stand for the operating system—Linux, Mac OS X, or Windows—plus the Apache Web server, the MySQL DBMS, and PHP.

TIP You'll be working with MySQL, so all the functions you use in this chapter are MySQL specific. For example, to connect to a database in MySQL the proper function is `mysql_connect()`, but if you're using PostgreSQL, you'd instead write `pg_connect()`. If you aren't using a MySQL DBMS, use the PHP manual (available through www.php.net) to find the appropriate function names.

MySQL Error Handling

Before this chapter gets too deep into working with MySQL, it would be best to discuss some error-handling techniques up front. Common errors you'll encounter are

- Failure to connect to MySQL
- Failure to select a database
- Inability to run a query
- No results returned by a query
- Data not inserted into a table

Experience will teach you why these errors normally occur, but immediately seeing what the problem is when running your scripts can save you much debugging time.

To have your scripts give informative reports about errors that occur, use the `mysql_error()` function. This function returns a textual version of the error that the MySQL server returned.

Along with this function, you may want to use some PHP tools for handling errors. Specifically, the error suppression operator (`@`), when used preceding a function name, suppresses any error messages or warnings the function might invoke:

```
@function_name();
```

Note that this operator doesn't stop the error from happening; it just prevents the message from being immediately displayed. You'd use it in situations where you intend to handle the error yourself, should one occur.

To use error handling:

1. Open `mysql_connect.php` (Script 12.1) in your text editor or IDE.
2. Suppress any PHP errors created by the `mysql_connect()` function by changing the `if` conditional as follows (Script 12.2):

```
if ($dbc = @mysql_connect
→ ('localhost', 'username',
→ 'password')) {
```

Rather than have PHP print out an error message when the `mysql_connect()` function backfires (ⓘ in the previous section), the message will be suppressed here using the `@` symbol. The errors still occur, but they're handled by the change made in the next step.

continues on next page

Script 12.2 By adding error control to the script (the `@` symbol and the `mysql_error()` function), you can more purposefully address problems that occur.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4  <head>
5    <meta http-equiv="content-type" content="text/html; charset=utf-8" />
6    <title>Connect to MySQL</title>
7  </head>
8  <body>
9  <?php // Script 12.2 - mysql_connect.php #2
10 /* This script connects to the MySQL server. */
11
12 // Attempt to connect to MySQL and print out messages:
13 if ($dbc = @mysql_connect('localhost', 'username', 'password')) {
14     print '<p>Successfully connected to MySQL!</p>';
15
16     mysql_close($dbc); // Close the connection.
17 } else {
18
19     print '<p style="color: red;">Could not connect to MySQL:<br />' . mysql_error() . '</p>';
20
21 }
22
23 ?>
24
25 </body>
26 </html>
```

Everywhere you see mysql type mysqli

3. Add the `mysql_error()` function to the `print` statement in the `else` section:

```
print '<p style="color: red;">Could  
→ not connect to MySQL:<br />' .  
→ mysql_error() . '</p>';
```

Instead of printing a message or relying on whatever error PHP kicks out (see [B](#) in the previous section), the script now prints the MySQL error within this context. You accomplish this by printing some HTML concatenated with the `mysql_error()` function.

You should note that the `mysql_error()` function, in this case, is not provided with the database connection—`$dbc`—as an argument, since no database connection was made.

4. Save the file and test it again in your Web browser [A](#).

If there was a problem, this result now looks better than what would have been shown previously. If the script connected, the result is like that shown in [A](#) in the previous section, because neither of the error-management tools is involved.

TIP In this chapter, error messages are revealed to assist in the debugging process. Live Web sites should not have this level of explicit error messages shown to the user.

TIP You can use the `@` symbol to suppress errors, notices, or warnings stemming from any function, not just a MySQL-related one. For example:

```
@include('./filename.php');
```

TIP You may also see code where `die()`, which is an alias for `exit()`, is called when a connection error occurs. The thinking is that since a database connection cannot be made, there's no point in continuing. In my opinion, that's too heavy-handed of an approach.



[A](#) Using PHP's error-control functions, you can adjust how errors are handled.

Creating and Selecting a Database

Before a PHP script can interact with a database, the database must first be selected. Of course, in order for you to select a database, it must exist. You can create a database using PHP, the MySQL client, `phpMyAdmin`, or any number of tools, as long as the MySQL hostname/username/password combination you are using has permission to do so.

Database permissions are a bit more complicated than file permissions, but you need to understand this: Different types of users can be assigned different database capabilities. For example, one DBMS user may be able to create new databases and delete existing ones (you may have dozens of databases in your DBMS), but a lower-level user may only be able to create and modify tables within a single database. The most basic user may just be able to read from, but not modify, tables.

If you're using PHP and MySQL for a live, hosted site, the hosting company will most likely give you the second type of access—control over a single database but not

the DBMS itself—and establish the initial database for you. If you're working on your own server or have administrative access, you should have the capability to create new users and databases.

To create a database with PHP, you use the `mysql_query()` function to execute a **CREATE DATABASE *dbname*** SQL command:

```
mysql_query('CREATE DATABASE somedb',  
→ $dbc);
```

Once you've done this, you can select the database using `mysql_select_db()`:

```
mysql_select_db('somedb', $dbc);
```

Note that you have to create a database only once, but it must always be selected before any other queries are run on it. In other words, some readers will need to perform the first step, but every reader must take the second step with every PHP script.

In this example, you'll create a new database and then select it. To repeat, creating a database requires that you have administrator access. If your Web host restricts your access, the hosting company should create the initial database for you upon request; you can just write the second part of this script, which selects the database.

To create and select a database:

1. Open `mysql_connect.php` (Script 12.2) in your text editor or IDE.
2. After the first `print` statement, create the new database, if necessary (Script 12.3):

```
if (@mysql_query('CREATE DATABASE
→ myblog', $dbc)) {
    print '<p>The database has been
→ created!</p>';
} else {
    print '<p style="color: red;">
→ Could not create the database
→ because:<br />' .
→ mysql_error($dbc) . '</p>';
}
```

If you need to create the database, use this construct to handle the task

I CREATED A DATABASE FOR YOU. It is called myblogXX. The XX is whatever student number you are.

Script 12.3 Creating a new database consists of three steps: connecting to the database, running a **CREATE DATABASE** query using the `mysql_query()` function, and then closing the connection.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4 <head>
5 <meta http-equiv="content-type" content="text/html; charset=utf-8" />
6 <title>Create the Database</title>
7 </head>
8 <body>
9 <?php // Script 12.3 - create_db.php
10 /* This script connects to the MySQL server. It also creates and selects the database. */
11
12 // Attempt to connect to MySQL and print out messages:
13 if ($dbc = @mysql_connect('localhost', 'username', 'password')) {
14
15     print '<p>Successfully connected to MySQL!</p>';
16
17     // Try to create the database:
18     if (@mysql_query('CREATE DATABASE myblog', $dbc)) {
19         print '<p>The database has been created!</p>';
20     } else { // Could not create it.
21         print '<p style="color: red;">Could not create the database because:<br />' .
22             mysql_error($dbc) . '</p>';
23     }
24 }
```

code continues on next page

START AT STEP 3.

Everywhere you see mysql type mysqli

cleanly and effectively. The query—**CREATE DATABASE myblog**—is run using the **mysql_query()** function. The **@** symbol is used to suppress any error messages, which are instead handled by **print** in conjunction with the **mysql_error()** function in the **else** clause.

Note that this invocation of **mysql_error()** can be provided with the specific database connection: **\$dbc**.

If the database has already been created for you, skip this step.

3. Attempt to select the database:

```
if (@mysql_select_db('myblog',
→ $dbc)) {
    print '<p>The database has been
→ selected!</p>';
} else {
    print '<p style="color: red;">
→ Could not select the database
→ because:<br />' .
→ mysql_error($dbc) . '</p>';
}
```

continues on next page

Script 12.3 continued

```
24 // Try to select the database:
25 if (@mysql_select_db('myblog', $dbc)) {
26     print '<p>The database has been selected!</p>';
27 } else {
28     print '<p style="color: red;">Could not select the database because:<br />' .
29     mysql_error($dbc) . '</p>';
29 }
30
31 mysql_close($dbc); // Close the connection.
32
33 } else {
34
35     print '<p style="color: red;">Could not connect to MySQL:<br />' . mysql_error() . '</p>';
36
37 }
38
39 ?>
40 </body>
41 </html>
```

Everywhere you see mysql type mysqli

This conditional has the same structure as that in Step 2. If PHP can select the database, a message is printed. If it can't select the database, the specific MySQL error will be displayed instead.

Every PHP script that runs queries on a database must connect to MySQL and select the database in order to work.

4. If you want, change the page title to reflect this script's new purpose:

`<title>Create the Database</title>`

5. Save your script as `create_db.php`, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **A** and **B**.

TIP You probably won't create databases with any frequency and may not normally do so using a PHP script. Still, this example demonstrates both how you execute simple queries using PHP as well as the SQL command needed to create a database.

TIP You haven't done so in these examples, but in general it's a good idea to set your database information—hostname, username, password, and database name—as variables or constants. Then you can plug them into the appropriate functions. By doing so, you can separate the database specifics from the functionality of the script, allowing you to easily port that code to other applications.

Successfully connected to MySQL!

The database has been created!

The database has been selected!

A If the database could be created and selected, you'll see this result in the Web browser.

//set the database access information as constants:

```
DEFINE ('DB_HOST', 'localhost');
DEFINE ('DB_USER', 'webstuXX');
DEFINE ('DB_PASSWORD', 'your password');
DEFINE ('DB_NAME', 'myblogXX');
$dbc = @mysqli_connect (DB_HOST,
DB_USER, DB_PASSWORD, DB_NAME)
OR die ('Could not connect to MySQL:
'.mysqli_connect_error() );
```

// Set the encoding...

```
mysqli_set_charset($dbc, 'utf8');
```

Successfully connected to MySQL!

Could not create the database because:
Access denied for user 'username'@'localhost' to database 'myblog'.

Could not select the database because:
Unknown database 'myblog'.

B If the MySQL user doesn't have the authority to create a database, you'll see a message like this. A similar result will occur if the user doesn't have permission to select the database.

Creating a Table

Once you’ve created and selected the initial database, you can begin creating individual tables in it. A database can consist of multiple tables, but in this simple example you’ll create one table in which all the chapter’s data will be stored.

To create a table in the database, you’ll use SQL—the language that databases understand. Because SQL is a lot like spoken English, the proper query to create a new table reads like this:

```
CREATE TABLE tablename (column1
→ definition, column2 definition,
→ etc.)
```

For each column, separated by commas, you first indicate the column name and then the column type. Common types are **TEXT**, **VARCHAR** (a variable number of characters), **DATETIME**, and **INT** (integer).

Because it’s highly recommended that you create a column that acts as the *primary key* (a column used to refer to each row), a simple **CREATE** statement could be

```
CREATE TABLE my_table (
id INT PRIMARY KEY,
information TEXT
)
```

A table’s primary key is a special column of unique values that is used to refer to the table’s rows. The database makes an index of this column in order to more quickly navigate through the table. A table can have only one primary key, which you normally set up as an automatically incremented column of integers. The first row has a key of 1, the second has a key of 2, and so forth. Referring back to the key always retrieves the values for that row.

You can visit the MySQL Web site for more information on SQL and column definitions. By following the directions in this section, though, you should be able to accomplish some basic database tasks. The table that you’ll create in this example is represented by **Table 12.2**.

In this example, you’ll create the database table that will be used to store information submitted via an HTML form. In the next section of the chapter, you’ll write the script that inserts the submitted data into the table created here.

TABLE 12.2 The entries Table

Column Name	Column Type
entry_id	Positive, non-null, automatically incrementing integer
title	Text up to 100 characters in length
entry	Text of any length
date_entered	A timestamp including both the date and the time the row was added

To create a new table:

1. Begin a new PHP document in your text editor or IDE, to be named `create_table.php` (Script 12.4):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="content-type"
    → content="text/html;
    → charset=utf-8" />
    <title>Create a Table</title>
</head>
<body>
```

Everywhere you see `mysql` type `mysqli`

Script 12.4 To create a database table, define the appropriate SQL statement and then invoke the `mysql_query()` function.

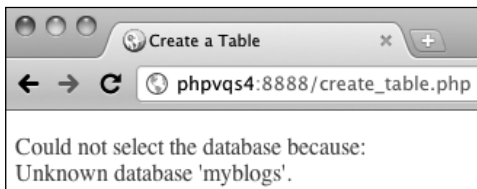
```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4  <head>
5  <meta http-equiv="content-type" content="text/html; charset=utf-8" />
6  <title>Create a Table</title>
7  </head>
8  <body>
9  <?php // Script 12.4 - create_table.php
10 /* This script connects to the MySQL server, selects the database, and creates a table. */
11
12 // Connect and select:
13 if ($dbc = @mysql_connect('localhost', 'username', 'password')) {
14
15     // Handle the error if the database couldn't be selected:
16     if (!@mysql_select_db('myblog', $dbc)) {
17         print '<p style="color: red;">Could not select the database because:<br />' .
18             mysql_error($dbc) . '<./p>';
19         mysql_close($dbc);
20         $dbc = FALSE;
21     }
22 } else { // Connection failure.
23     print '<p style="color: red;">Could not connect to MySQL:<br />' . mysql_error() . '<./p>';
24 }
```

code continues on next page

USE phpMyAdmin to create the table in your my_blog database.

Script 12.4 *continued*

```
25
26 if ($dbc) {
27
28     // Define the query:
29     $query = 'CREATE TABLE entries (
30     entry_id INT UNSIGNED NOT NULL
31     AUTO_INCREMENT PRIMARY KEY,
32     title VARCHAR(100) NOT NULL,
33     entry TEXT NOT NULL,
34     date_entered DATETIME NOT NULL
35 );';
36
37 // Execute the query:
38 if (@mysql_query($query, $dbc)) {
39     print '<p>The table has been
40     created!</p>';
41 } else {
42     print '<p style="color: red;">
43     Could not create the table
44     because:<br />' .
45     mysql_error($dbc) . '</p>
46     <p>The query being run was: ' .
47     $query . '</p>';
48 }
49
50 mysql_close($dbc); // Close the
51 connection.
52
53 }
54 ?>
55 </body>
56 </html>
```



A Between the MySQL error message and printing out the query being executed, you should be able to figure out what the problem is if the script does not work properly.

2. Begin a section of PHP code:

```
<?php // Script 12.4 -
→ create_table.php
```

3. Connect to the MySQL server, and select the database:

```
if ($dbc = @mysql_connect
→ ('localhost', 'username',
→ 'password')) {
    if (!@mysql_select_db('myblog',
→ $dbc)) {
        print '<p style="color:
→ red;">Could not select the
→ database because:<br />' .
        mysql_error($dbc) . '</p>';
        mysql_close($dbc);
        $dbc = FALSE;
    }
} else {
    print '<p style="color:
→ red;">Could not connect to
→ MySQL:<br />' . mysql_error()
→ . '</p>';
}
```

This is an alternative version of the code used in the preceding script. The main difference is that no messages are printed if each step was successful (hopefully you have connection and database selection working by this point).

If for some reason a connection could not be made to the database, or the database could not be selected, then error messages will be displayed **A**. If a connection could be made but the database could not be selected, the connection is then closed, as the connection won't be useful. To indicate that scenario, the `$dbc` variable, which had represented the connection, is set to **FALSE**, which will prevent the **CREATE** query from being executed (see Step 4).

continues on next page

READ FOR YOUR INFORMATION ONLY. We are going create the table in phpMyAdmin

4. Define the query for creating the table:

```
if ($dbc) {  
    $query = 'CREATE TABLE entries (  
        entry_id INT UNSIGNED NOT NULL  
        → AUTO_INCREMENT PRIMARY KEY,  
        title VARCHAR(100) NOT NULL,  
        entry TEXT NOT NULL,  
        date_entered DATETIME NOT NULL  
    )';
```

First, if `$dbc` still has a value, the table can be created. If not—meaning that no connection could be made or the database couldn't be selected—then none of the following code will be executed.

As for the query itself, let's break that into more recognizable parts. First, to create a new table, you write **CREATE TABLE *tablename*** (where *tablename* is replaced by the desired table name). Then, within parentheses, you list every column you want with each column separated by a comma. Your table and column names should be alphanumeric, with no spaces.

The first column in the table is called **entry_id**; it's an unsigned integer (**INT UNSIGNED**—which means that it can only be a positive whole number). By including the words **NOT NULL**, you indicate that this column must have a value for each row. The values automatically increase by 1 for each

row added (**AUTO INCREMENT**) and stand as the primary key.

The next two columns consist of text. One, called **title**, is limited to 100 characters. The second, **entry**, can be vast in size. Each of these fields is also marked as **NOT NULL**, making them required fields.

Finally, the **date_entered** column is a timestamp that marks when each record was added to the table.

5. Execute the query:

```
if (@mysql_query($query, $dbc)) {  
    print '<p>The table has been  
        → created.</p>';  
} else {  
    print '<p style="color:  
        → red;">Could not create the  
        → table because:<br />' .  
        → mysql_error($dbc) .  
        → '</p><p>The query being run  
        → was: ' . $query . '</p>';  
}
```

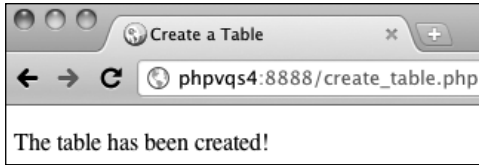
To create the table, call the `mysql_query()` function using the `$query` variable as the first argument and the database connection as the second. If a problem occurred, the MySQL error is printed, along with the value of the `$query` variable. This last step—printing the actual query being executed—is a particularly useful debugging technique **B**.

Could not create the table because:
CREATE command denied to user 'username'@'localhost' for table 'entries'.

The query being run was: CREATE TABLE entries (entry_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY, title VARCHAR(100) NOT NULL, entry TEXT NOT NULL, date_entered DATETIME NOT NULL)

B If the query caused an error, the MySQL error will be reported and the query itself displayed (for debugging purposes).

READ FOR YOUR INFORMATION ONLY. We are not going to create a php page. We will use phpMyAdmin



❶ If all went well, all you'll see is this message.

6. Close the database connection and complete the `$dbc` conditional:

```
mysql_close($dbc);
}
```
7. Complete the PHP code and the HTML page:

```
?>
</body>
</html>
```
8. Save the script as **create_table.php**, place it in the proper directory for your PHP-enabled server, and test it in your Web browser ❷.

TIP It's not necessary to write your SQL keywords in all capital letters as I do here, but doing so helps to distinguish the SQL terms from the table and column names.

TIP On larger Web applications, I highly recommended that you place the database connection and selection code (lines 13 through 24 here) in a separate file, located outside of the Web directory. Then, each page that requires the database can include this external file. You'll see an example of this in Chapter 13, "Putting It All Together."

TIP The `mysql_query()` function returns TRUE if a query was successfully run on a database. That result doesn't necessarily mean the desired result occurred.

TIP This chapter presents the basics of MySQL- and SQL-related knowledge (including column types). You'll want to check out other resources—listed in Appendix B—once you're comfortable with the fundamentals.

TIP You wouldn't normally use a PHP script to create a table, just as you wouldn't normally create a database using a PHP script, but when you're just starting with MySQL, this is an easy way to achieve the desired results.

This is why we are using phpMyAdmin



Inserting Data into a Database

As mentioned, this database will be used as a simple blog, an online journal. Blog entries—consisting of a title and text—will be added to the database using one page and then displayed on another page.

With the last script, you created the table, which consists of four columns: **entry_id**, **title**, **entry**, and **date_entered**. The process of adding information to a table is similar to creating the table itself in terms of which PHP functions you use, but the SQL query is different. To insert records, use the **INSERT** SQL command with either of the following syntaxes:

```
INSERT INTO tablename VALUES (value1,  
→ value2, value3, etc.)  
INSERT INTO tablename (column1_name,  
→ column2_name) VALUES (value1, value2)
```

The query begins with **INSERT INTO** *tablename*. Then you can either specify which columns you're inserting values for or not. The latter is more specific and is therefore preferred, but it can be tedious

if you're populating a slew of columns. In either case, you must be certain to list the right number and type of values for each column.

The values are placed within parentheses, with each value separated by a comma. Non-numeric values—strings and dates—need to be quoted, whereas numbers do not:

```
INSERT INTO example (name, age)  
→ VALUES ('Jonah', 1)
```

The query is executed using the **mysql_query()** function. Because **INSERT** queries can be complex, it makes sense to assign each query to a variable and send that variable to the **mysql_query()** function (as previously demonstrated).

To demonstrate, let's create a page that adds blog entries to the database. Like many of the examples in the preceding chapter, this one will both display and handle the HTML form. Before getting into the example, though, I'll say that this script knowingly has a security hole in it; it'll be explained and fixed in the next section of the chapter.

Building on This Example

The focus in this chapter is on explaining and demonstrating the basics of using PHP with MySQL. This also includes the core components of SQL. However, this chapter's examples do a few things that you wouldn't want to do in a real site, such as allow anyone to insert, edit, and delete database records.

In the next chapter, a different example will be developed that is also database driven. That example will use cookies to restrict what users can do with the site.

To enter data into a database from an HTML form:

1. Begin a new PHP document in your text editor or IDE, to be named `add_entry.php` (Script 12.5):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
→ "http://www.w3.org/TR/xhtml1/
→ DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
  <meta http-equiv="content-type"
→ content="text/html;
→ charset=utf-8" />
  <title>Add a Blog Entry</title>
</head>
<body>
<h1>Add a Blog Entry</h1>
```

continues on next page

Script 12.5 The query statement for adding information to a database is straightforward enough, but be sure to match the number of values in parentheses to the number of columns in the database table.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4 <head>
5   <meta http-equiv="content-type" content="text/html; charset=utf-8" />
6   <title>Add a Blog Entry</title>
7 </head>
8 <body>
9 <h1>Add a Blog Entry</h1>
10 <?php // Script 12.5 - add_entry.php
11 /* This script adds a blog entry to the database. */
12
13 if ($_SERVER['REQUEST_METHOD'] == 'POST') { // Handle the form.
14
15   // Connect and select:
16   $dbc = mysql_connect('localhost', 'username', 'password');
17   mysql_select_db('myblog', $dbc);
18
19   // Validate the form data:
20   $problem = FALSE;
21   if (!empty($_POST['title']) && !empty($_POST['entry'])) {
```

Use a `require()` construct here instead of writing this script for each page that connects to the database.

code continues on next page

`require('mysqli_connect.php');` //Connect to the db.

2. Create the initial PHP section and check for the form submission:

```
<?php // Script 12.5 -  
→ add_entry.php  
if ($_SERVER['REQUEST_METHOD'] ==  
→ 'POST') {
```

3. Connect to and select the database:

```
$dbc = mysql_connect('localhost',  
→ 'username', 'password');  
mysql_select_db('myblog', $dbc);
```

At this point, if you're running these examples in order, I'll assume you have a working connection and selection process down, so I'll dispense with all the conditionals and error reporting (mostly to shorten the script). If you have problems connecting to and selecting the database, apply the code already outlined in the chapter.

`require('mysqli_connect.php'); //Connect to the db.`

Script 12.5 *continued*

```
22     $title = trim(strip_tags($_POST['title']));  
23     $entry = trim(strip_tags($_POST['entry']));  
24     } else {  
25         print '<p style="color: red;">Please submit both a title and an entry.</p>';  
26         $problem = TRUE;  
27     }  
28  
29     if (!$problem) {  
30  
31         // Define the query:  
32         $query = "INSERT INTO entries (entry_id, title, entry, date_entered) VALUES  
                 (0, '$title', '$entry', NOW())";  
33  
34         // Execute the query:  
35         if (@mysql_query($query, $dbc)) {  
36             print '<p>The blog entry has been added!</p>';  
37         } else {  
38             print '<p style="color: red;">Could not add the entry because:<br /> ' .  
                 mysql_error($dbc) . ' .</p><p>The query being run was: ' . $query . ' .</p>';  
39         }  
40  
41     } // No problem!  
42  
43     mysql_close($dbc); // Close the connection.  
44  
45 } // End of form submission IF.  
46  
47 // Display the form:  
48 ?>  
49 <form action="add_entry.php" method="post">  
50     <p>Entry Title: <input type="text" name="title" size="40" maxsize="100" /></p>  
51     <p>Entry Text: <textarea name="entry" cols="40" rows="5"></textarea></p>  
52     <input type="submit" name="submit" value="Post This Entry!" />  
53 </form>  
54 </body>  
55 </html>
```

Use TIP on page 357

Add a Blog Entry

Please submit both a title and an entry.

Entry Title:

Entry Text:

A The PHP script performs some basic form validation so that empty records are not inserted into the database.

Use TIP from page 357
 "INSERT INTO entries (title,
 entry, date_entered)
 VALUES ('\$title', '\$entry',
 NOW());"

4. Validate the form data:

```
$problem = FALSE;
if (empty($_POST['title']) &&
    !empty($_POST['entry'])) {
    $title = trim(strip_tags
        → ($_POST['title']));
    $entry = trim(strip_tags
        → ($_POST['entry']));
} else {
    print '<p style="color:
        → red;">Please submit both a
        → title and an entry.</p>';
    $problem = TRUE;
}
```

Before you use the form data in an **INSERT** query, it ought to be validated. Just a minimum of validation is used here, guaranteeing that some values are provided. If so, new variables are assigned those values, after trimming away extraneous spaces and applying **strip_tags()** (to prevent cross-site scripting attacks and other potential problems). If either of the values was empty, an error message is printed **A** and the **\$problem** flag variable is set to **TRUE** (because there is a problem).

5. Define the **INSERT** query:

```
if (!$problem) {
    $query = "INSERT INTO entries
        → (entry_id, title, entry,
        → date_entered) VALUES
        → (o, '$title', '$entry', NOW());"
```

The query begins with the necessary **INSERT INTO tablename** code. Then it lists the columns for which values will be submitted. After that is **VALUES**, followed by four values (one for each column, in order) separated by commas. When assigning this query to the **\$query** variable, use double quotation marks so that the values of the variables will

continues on next page

REMEMBER: We are using mysqli

be automatically inserted by PHP. The `$title` and `$entry` variables are strings, so they must be placed within single quotation marks in the query itself.

Because the `entry_id` column has been set to `AUTO_INCREMENT`, you can use 0 as the value and MySQL will automatically use the next logical value for that column. To set the value of the `date_entered` column, use the MySQL `NOW()` function. It inserts the current time as that value.

6. Run the query on the database:

```
if (@mysqli_query($query, $dbc)) {  
    print '<p>The blog entry has  
    → been added!</p>';  
} else {  
    print '<p style="color: red;">  
    → Could not add the entry  
    → because:<br />' . mysqli_  
    → error($dbc) . '</p><p>The  
    → query being run was: ' .  
    → $query . '</p>';  
}
```

The query, once defined, is run using the `mysqli_query()` function. By calling this function as the condition of an `if-else` statement, you can print simple messages indicating the result of the query execution.

As an essential debugging tool, if the query didn't run properly, the MySQL error and the query being run are both printed to the Web browser **B**.

7. Close the `$problem` conditional, the database connection, and complete the main conditional and the PHP section:

```
} // No problem!  
mysqli_close($dbc);  
} // End of form submission IF.  
?>
```

From here on out, the form will be displayed.

8. Create the form:

```
<form action="add_entry.php"  
→ method="post">  
    <p>Entry Title: <input type=  
    → "text" name="title" size="40"  
    → maxsize="100" /></p>  
    <p>Entry Text: <textarea  
    → name="entry" cols="40"  
    → rows="5"></textarea></p>  
    <input type="submit" name=  
    → "submit" value="Post This  
    → Entry!" />  
</form>
```

Add a Blog Entry

Could not add the entry because:

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 's January 1st, 2011, the first day of six weeks of writing the date incorrectly.' at line 1.

The query being run was: INSERT INTO entries (entry_id, title, entry, date_entered) VALUES (0, 'Happy New Year!', 'It's January 1st, 2011, the first day of six weeks of writing the date incorrectly.', NOW())

B If the `INSERT` query didn't work, the MySQL error is printed along with the query that was run.

TEST!!!!

A screenshot of a web browser window titled 'Add a Blog Entry'. The address bar shows 'phpvqs4:8888/add_entry.php'. The page has a heading 'Add a Blog Entry'. Below it, there is an 'Entry Title:' label followed by a text input field containing 'Happy New Year!'. Underneath is a text area with the placeholder text 'It's January 1st, 2011, the first day of six weeks of writing the date incorrectly.' and an 'Entry Text:' label. At the bottom is a button labeled 'Post This Entry!'.

C This is the form for adding an entry to the database.

A screenshot of the same web page after a successful database insert. The heading 'Add a Blog Entry' is still present. Below it is a message 'The blog entry has been added!'. Underneath is an 'Entry Title:' label followed by an empty text input field.

D If the **INSERT** query ran properly, a message is printed and the form is displayed again.

The HTML form is very simple, requiring only a title for the blog entry and the entry itself. As a good rule of thumb, use the same name for your form inputs as the corresponding column names in the database. Doing so makes errors less likely.

9. Finish the HTML page:

```
</body>
</html>
```

10. Save the script as **add_entry.php**, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **C** and **D**.

You should probably avoid using apostrophes in your form values or you might see results like those in **B**. Turn the page for the explanation and solution.

TIP MySQL allows you to insert several records at once, using this format:

```
INSERT INTO tablename (column1_name,
→ column2_name) VALUES (value1,
→ value2), (value3, value4);
```

Most other database applications don't support this construct, though.

TIP To retrieve the automatically incremented number created for an **AUTO_INCREMENT** column, call the `mysql_insert_id()` function.

TIP Because of the way auto-incrementing primary keys work, this query is also fine:

```
INSERT INTO entries (title, entry,
→ date_entered) VALUES ('$title',
→ '$entry', NOW());
```

We are using this
in our script.



Securing Query Data

As I mentioned in the introduction to the preceding sequence of steps, the code as written has a pretty bad security hole in it. As it stands, if someone submits text that contains an apostrophe, that data will break the SQL query ^A (security concerns aside, it's also a pretty bad bug). The result is obviously undesirable, but why is it insecure?

If a malicious user knows they can break a query by typing an apostrophe, they may try to run their own queries using this hole. If someone submitted `';DROP TABLE entries;` as the blog post title, the resulting query would be

```
INSERT INTO entries (entry_id,
→ title, entry, date_entered) VALUES
→ (0, '';DROP TABLE entries;', '<entry
→ text>', NOW())
```

The initial apostrophe in the provided entry title has the effect of completing the blog title value part of the query. The semicolon then terminates the **INSERT** query itself. This will make the original query syntactically invalid. Then the database will be provided with a second query—**DROP TABLE entries**, with the hope that it will be executed when the original **INSERT** query fails. This is called an *SQL injection attack*, but fortunately it's easy to prevent.

To do so, send potentially insecure data to be used in a query through the `mysql_real_escape_string()` function. This function will escape—preface with a backslash—any potentially harmful characters, making the data safe to use in a query:

```
$var = mysql_real_escape_string
→ ($var, $dbc);
```

Let's apply this function to the preceding script.

Add a Blog Entry

Could not add the entry because:

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 `';DROP TABLE entries;', ''`, `NOW()`' at line 1.

The query being run was: `INSERT INTO entries (entry_id, title, entry, date_entered) VALUES (0, '';DROP TABLE entries;', ''`, `NOW()`)

^A The apostrophe in the conjunction *It's* breaks the query because apostrophes (or single quotation marks) are used to delimit strings used in queries.

Showing MySQL Errors

Even if MySQL doesn't execute an injected SQL command (normally MySQL will only run a single SQL query sent through the `mysql_query()` function), hackers will provide bad characters in form data in the hopes that the syntactically broken query generates a database error. By seeing the database error, the hacker seeks to gain knowledge about the database that can be used for malicious purposes. For this reason, it's imperative that a live site never reveal the actual MySQL errors or queries being executed. The scripts in this chapter do so only for your own debugging purposes.

```
$title = mysqli_real_escape_string($dbc, trim(strip_tags($_POST['title'])) );  
$entry = mysqli_real_escape_string($dbc, trim(strip_tags($_POST['entry'])) );
```

To secure query data:

1. Open **add_entry.php** (Script 12.5) in your text editor or IDE, if it is not already open.
2. Update the assignment of the **\$title** and **\$entry** variables to read (Script 12.6) as follows:

```
$title = mysqli_real_escape_string  
→ (trim(strip_tags($_POST  
→ ['title']))), $dbc);  
$entry = mysqli_real_escape_string  
→ (trim(strip_tags($_POST  
→ ['entry']))), $dbc);
```

These two lines will greatly improve the security and functionality of the script. For both posted variables, their values are first trimmed and stripped of tags, then sent through **mysqli_real_escape_string()**. The result will be safe to use in the query.

If the application of three functions to one variable is too confusing for you, you can separate the code into discrete steps:

```
$title = $_POST['title'];  
$title = trim(strip_tags($title));  
$title = mysqli_real_escape_  
string($title, $dbc);
```

continues on next page

Script 12.6 To better secure the Web application and the database, the **mysqli_real_escape_string()** function is applied to the form data used in the query.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">  
3 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">  
4 <head>  
5 <meta http-equiv="content-type" content="text/html; charset=utf-8" />  
6 <title>Add a Blog Entry</title>  
7 </head>  
8 <body>  
9 <h1>Add a Blog Entry</h1>  
10 <?php // Script 12.6 - add_entry.php #2  
11 /* This script adds a blog entry to the database. It now does so securely! */  
12  
13 if (isset($_POST['submitted'])) { // Handle the form.  
14  
15 // Connect and select:  
16 $dbc = mysqli_connect('localhost', 'username', 'password');  
17 mysqli_select_db('myblog', $dbc);  
18  
19 // Validate and secure the form data:  
20 $problem = FALSE;  
21 if (!empty($_POST['title']) && !empty($_POST['entry'])) {  
22 $title = mysqli_real_escape_string(trim(strip_tags($_POST['title'])), $dbc);  
23 $entry = mysqli_real_escape_string(trim(strip_tags($_POST['entry'])), $dbc);  
24 } else {  
25 print '<p style="color: red;">Please submit both a title and an entry.</p>';  
26 $problem = TRUE;  
27 }  
28
```

The order of this function parameters changed in mysqli. Please see correction at the top of the page.

code continues on next page

Everywhere you see mysql type mysqli

3. Save the script, place it on your PHP-enabled server, and test it in your Web browser **B** and **C**.

TIP If you see (later in the chapter) that the displayed blog posts have extra backslashes before apostrophes, this is likely because you're using a version of PHP with Magic Quotes enabled. (Magic Quotes automatically escapes problematic characters in form data, although not as well as `mysql_real_escape_string()`). If that's the case, you'll need to apply the `stripslashes()` function to remove the extraneous slashes from the submitted values:

```
$title = mysql_real_escape_string  
→ (stripslashes(trim(strip_tags  
→ ($_POST['title']))), $dbc);
```

MAGIC
QUOTES
DEPRECATED

Script 12.6 *continued*

```
29     if (!$problem) {  
30  
31         // Define the query:  
32         $query = "INSERT INTO entries (entry_id, title, entry, date_entered) VALUES (0, '$title',  
           'entry', NOW())";  
33  
34         // Execute the query:  
35         if (@mysql_query($query, $dbc)) {  
36             print "<p>The blog entry has been added!</p>";  
37         } else {  
38             print "<p style='color: red;'>Could not add the entry because:<br /> ' .  
           mysql_error($dbc) . '</p><p>The query being run was: ' . $query . '</p>";  
39         }  
40  
41     } // No problem!  
42  
43     mysql_close($dbc); // Close the connection.  
44  
45 } // End of form submission IF.  
46  
47 // Display the form:  
48 ?>  
49 <form action="add_entry.php" method="post">  
50     <p>Entry Title: <input type="text" name="title" size="40" maxsize="100" /></p>  
51     <p>Entry Text: <textarea name="entry" cols="40" rows="5"></textarea></p>  
52     <input type="submit" name="submit" value="Post This Entry!" />  
53     <input type="hidden" name="submitted" value="true" />  
54 </form>  
55 </body>  
56 </html>
```

Add a Blog Entry

Entry Title:

"Will these quotes and apostrophes cause problems?",
you ask. I don't think so!

Entry Text:

- B** Now apostrophes in form data...

Add a Blog Entry

The blog entry has been added!

- C** ...will not cause problems.

Retrieving Data from a Database

The next process this chapter demonstrates for working with databases is retrieving data from a populated table. You still use the `mysqli_query()` function to run the query, but retrieving data is slightly different than inserting data—you have to assign the query result to a variable and then use another function in order to fetch the data.

The basic syntax for retrieving data is the **SELECT** query:

```
SELECT what columns FROM what table
```

The easiest query for reading data from a table is

```
SELECT * FROM tablename
```

The asterisk is the equivalent of saying *every column*. If you only require certain columns to be returned, you can limit your query, like so:

```
SELECT name, email FROM users
```

This query requests that only the information from two columns (**name** and **email**) be gathered. Keep in mind that this structure doesn't limit what rows (or records) are returned, just what columns for those rows.

Another way to alter your query is to add a conditional restricting which rows are returned, accomplished using a **WHERE** clause:

```
SELECT * FROM users WHERE  
→ name='Larry'
```

Here you want the information from every column in the table, but only from the rows where the **name** column is equal to *Larry*.

This is a good example of how SQL uses only a few terms effectively and flexibly.

The main difference in retrieving data from a database as opposed to inserting data into a database is that you need to handle the query differently. You should first assign the results of the query to a variable:

```
$result = mysqli_query($query, $dbc);
```

Just as `$dbc` is a reference to an open database connection, `$result` is a reference to a query result set. This variable is then provided to the `mysqli_fetch_array()` function, which retrieves the query results:

```
$row = mysqli_fetch_array($result);
```

The function fetches one row from the result set at a time, creating an array in the process. The array will use the selected column names as its indexes: `$row['name']`, `$row['email']`, and so on. As with any array, you must refer to the columns exactly as they're defined in the database (the keys are case-sensitive). So, in this example, you must use `$row['email']` instead of `$row['Email']`.

If the query will return multiple rows, execute the `mysqli_fetch_array()` function within a loop to access them all:

```
while ($row = mysqli_fetch_array  
→ ($result)) {  
    // Do something with $row.  
}
```

With each iteration of the loop, the next row of information from the query (referenced by `$result`) is assigned to an array called `$row`. This process continues until no more rows of information are found. Within the loop, you would do whatever you want with `$row`.

continues on next page

The best way to understand this new code is to try it. You'll write a script that retrieves the posts stored in the **entries** table and displays them **A**. You may want to run through **add_entry.php** a couple more times to build up the table first.

To retrieve data from a table:

1. Begin a new PHP document in your text editor or IDE, to be named **view_entries.php** (Script 12.7):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="content-type"
    → content="text/html;
    → charset=utf-8" />
    <title>View My Blog</title>
</head>
<body>
    <h1>My Blog</h1>
```

My Blog

It's Another Test!

"Will these quotes and apostrophes cause problems?", you ask. I don't think so!

[Edit](#) [Delete](#)

Happy New Year!

Today is January 1st, 2011, the first day of six weeks of writing the date incorrectly.

[Edit](#) [Delete](#)

A This dynamic Web page uses PHP to pull data from a database.

Script 12.7 The SQL query for retrieving all data from a table is quite simple; but in order for PHP to access every returned record, you must loop through the results one row at a time.

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4  <head>
5      <meta http-equiv="content-type" content="text/html; charset=utf-8" />
6      <title>View My Blog</title>
7  </head>
8  <body>
9      <h1>My Blog</h1>
10  <?php // Script 12.7 - view_entries.php
11  /* This script retrieves blog entries from the database. */
12
```

code continues on next page

2. Begin a PHP section and connect to the database:

```
<?php // Script 12.7 -
→ view_entries.php
$dbc = mysql_connect('localhost',
→ 'username', 'password');
mysql_select_db('myblog', $dbc);
```

3. Define the **SELECT** query:

```
$query = 'SELECT * FROM entries
→ ORDER BY date_entered DESC';
```

This basic query tells the database that you'd like to fetch every column of every row in the **entries** table. The returned records should be sorted, as indicated by the **ORDER BY** clause, by

the order in which they were entered (recorded in the **date_entered** column), starting with the most recent first. This last option is set by **DESC**, which is short for *descending*. If the query was **ORDER BY date_entered ASC**, the most recently added record would be retrieved last.

4. Run the query:

```
if ($r = mysql_query($query,
→ $dbc)) {
```

The **SELECT** query is run like any other. However, the result of the query is assigned to a **\$result** (or, more tersely, **\$r**) variable, which will be referenced later.

continues on next page

Script 12.7 *continued*

```
13 // Connect and select:
14 $dbc = mysql_connect('localhost', 'username', 'password');
15 mysql_select_db('myblog', $dbc);
16
17 // Define the query:
18 $query = 'SELECT * FROM entries ORDER BY date_entered DESC';
19
20 if ($r = mysql_query($query, $dbc)) { // Run the query.
21
22     // Retrieve and print every record:
23     while ($row = mysql_fetch_array($r)) {
24         print "<p><h3>{$row['title']}</h3>";
25         {$row['entry']}<br />
26         <a href="edit_entry.php?id={$row['entry_id']}">Edit</a>
27         <a href="delete_entry.php?id={$row['entry_id']}">Delete</a>
28         </p><hr />\n";
29     }
30
31 } else { // Query didn't run.
32     print '<p style="color: red;">Could not retrieve the data because:<br />' . mysql_error($dbc)
33     . '</p><p>The query being run was: ' . $query . '</p>';
34 } // End of query IF.
35 mysql_close($dbc); // Close the connection.
36
37 ?>
38 </body>
39 </html>
```

Use a **require** construct here instead of writing this script for each page that connects to the database.

5. Print out the returned results:

```
while ($row = mysql_fetch_array
→ ($r)) {
    print "<p><h3>{$row['title']}</h3>
    {$row['entry']}<br />
    <a href=\"edit_entry.php?id=
    → {$row['entry_id']}\">Edit</a>
    <a href=\"delete_entry.php?
    → id={$row['entry_id']}\">
    → Delete</a>
    </p><hr />\n";
}
```

This loop sets the variable **\$row** to an array containing the first record returned in **\$r**. The loop then executes the following command (the **print** statement). Once the loop gets back to the beginning, it assigns the next row, if it exists. It continues to do this until there are no more rows of information to be obtained.

Within the loop, the array's keys are the names of the columns from the table—hence, **entry_id**, **title**, and **entry** (there's no need to print out the **date_entered**).

At the bottom of each post, two links are created: to **edit_entry.php** and **delete_entry.php**. These scripts will be written in the rest of the chapter. Each link passes the posting's database ID value along in the URL. That information will be necessary for those other two pages to edit and delete the blog posting accordingly.

6. Handle the errors if the query didn't run:

```
} else { // Query didn't run.
    print '<p style="color: red;">
    → Could not retrieve the
    → data because:<br />' .
    → mysql_error($dbc) . '</p>
    → <p>The query being run
    → was: ' . $query . '</p>';
} // End of query IF.
```


My Blog

This is the newest post!

This is so absolutely amazing that I'm downright speechless!
[Edit](#) [Delete](#)

It's Another Test!

"Will these quotes and apostrophes cause problems?", you ask. I don't think so!
[Edit](#) [Delete](#)

Happy New Year!

B Thanks to the **SELECT** query, which orders the returned records by the date they were entered, the most recently added entry is always listed first.

```
<h1>My Blog</h1>
<p><h3>This is the newest post!</h3>
    This is so absolutely amazing that I'm down
    <a href="edit_entry.php?id=4">Edit</a>
    <a href="delete_entry.php?id=4">Delete</a>
</p><hr />
<p><h3>It's Another Test!</h3>
    "Will these quotes and apostrophes cause p
    <a href="edit_entry.php?id=3">Edit</a>
    <a href="delete_entry.php?id=3">Delete</a>
</p><hr />
<p><h3>Happy New Year!</h3>
    Today is January 1st, 2011, the first day
    incorrectly.<br />
    <a href="edit_entry.php?id=2">Edit</a>
    <a href="delete_entry.php?id=2">Delete</a>
</p><hr />
```

C Part of the HTML source of the page. Note that the two links have `?id=X` appended to each URL.

If the query couldn't run on the database, it should be printed out, along with the MySQL error (for debugging purposes).

7. Close the database connection:

```
mysqli_close($dbc);
```

8. Complete the PHP section and the HTML page:

```
?>
</body>
</html>
```

9. Save the script as **view_entries.php**, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **A**.

10. If you want, add another record to the blog using the **add_entry.php** page (Script 12.6), and run this page again **B**.

11. Check the source code of the page to see the dynamically generated links **C**.

TIP The `mysqli_fetch_array()` function takes another argument, which is a constant indicating what kind of array should be returned. `MYSQL_ASSOC` returns an associative array, whereas `MYSQL_NUM` returns a numerically indexed array.

TIP The `mysqli_num_rows()` function returns the number of records returned by a **SELECT** query.

TIP It's possible to paginate returned records so that 10 or 20 appear on each page (like the way Google works). Doing so requires more advanced coding than can be taught in this book, though. See my book *PHP 6 and MySQL 5 for Dynamic Web Sites: Visual QuickPro Guide* (Peachpit Press, 2007), or look online for code examples and tutorials.

Deleting Data in a Database

Sometimes you might also want to run a **DELETE** query on a database. Such a query removes records from the database. The syntax for a delete query is

DELETE FROM *tablename* WHERE
→ ***column=value***

The **WHERE** clause isn't required, but if it's omitted, you'll remove every record from the table. You should also understand that once you delete a record, there's no way to recover it (unless you have a backup of the database).

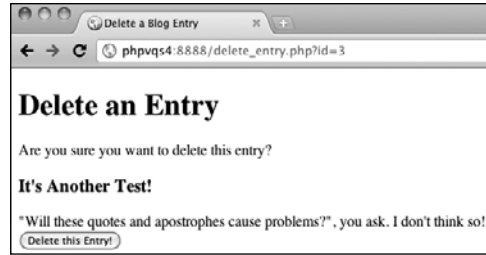
As a safeguard, if you want to delete only a single record from a table, add the **LIMIT** clause to the query:

DELETE FROM *tablename* WHERE
→ ***column=value* LIMIT 1**

This clause ensures that only one record is deleted at most. Once you've defined your query, it's again executed using the **mysql_query()** function, like any other query.

To see if a **DELETE** query worked, you can use the **mysql_affected_rows()** function. This function returns the number of rows affected by an **INSERT**, **DELETE**, or **UPDATE** query.

As an example, let's write the **delete_entry.php** script, which is linked from the **view_blog.php** page. This page receives the database record ID in the URL. It then displays the entry to confirm that the user wants to delete it **A**. If the user clicks the button, the record will be deleted **B**.



A When the user arrives at this page, the blog entry is shown and the user must confirm that they want to delete it.



B If the delete query worked properly, the user sees this result.

**LET'S BACKUP
OUR DATABASE
BEFORE WE
MAKE ANY
CHANGES!!!
Export out a copy
and save it to your
desktop.**

To delete data from a database:

1. Begin a new PHP document in your text editor or IDE, to be named **delete_entry.php (Script 12.8):**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
→ XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/
    → DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="content-type"
    → content="text/html;
    → charset=utf-8" />
    <title>Delete a Blog Entry
    → </title>
</head>
<body>
<h1>Delete an Entry</h1>
```

2. Start the PHP code and connect to the database:

```
<?php // Script 12.8 -
→ delete_entry.php
$dbc = mysql_connect('localhost',
→ 'username', 'password');
mysql_select_db('myblog', $dbc);
```

```
require('mysqli_connect.php');
//Connect to the db.
```

continues on page 369


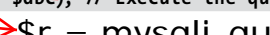
Script 12.8 The **DELETE** SQL command permanently removes a record (or records) from a table.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4 <head>
5     <meta http-equiv="content-type" content="text/html; charset=utf-8" />
6     <title>Delete a Blog Entry</title>
7 </head>
8 <body>
9 <h1>Delete an Entry</h1>
10 <?php // Script 12.8 - delete_entry.php
11 /* This script deletes a blog entry. */
12
```

code continues on next page

REMEMBER: We are using mysqli

Script 12.8 *continued*

```
13 // Connect and select:  require('mysqli_connect.php');
14 $dbc = mysqli_connect('localhost', 'username', 'password'); //Connect to the db.
15 mysql_select_db('myblog', $dbc);
16
17 if (isset($_GET['id']) && is_numeric($_GET['id']) ) { // Display the entry in a form:
18
19     // Define the query:
20     $query = "SELECT title, entry FROM entries WHERE entry_id={$_GET['id']}";
21     if ($r = mysqli_query($query, $dbc)) { // Run the query.
22
23         $row = mysqli_fetch_array($r); // Retrieve the information.
24
25         // Make the form:
26         print '<form action="delete_entry.php" method="post">
27         <p>Are you sure you want to delete this entry?</p>
28         <p><h3>' . $row['title'] . '</h3>' .
29         $row['entry'] . '<br />
30         <input type="hidden" name="id" value="' . $_GET['id'] . '" />
31         <input type="submit" name="submit" value="Delete this Entry!" /></p>
32         </form>';
33
34     } else { // Couldn't get the information.
35         print '<p style="color: red;">Could not retrieve the blog entry because:<br />' .
36         mysqli_error($dbc) . '</p><p>The query being run was: ' . $query . '</p>';
37     }
38 } elseif (isset($_POST['id']) && is_numeric($_POST['id'])) { // Handle the form.
39
40     // Define the query:
41     $query = "DELETE FROM entries WHERE entry_id={$_POST['id']} LIMIT 1";
42     $r = mysqli_query($query, $dbc); // Execute the query.
43      $r = mysqli_query($dbc, $query);
44     // Report on the result:
45     if (mysqli_affected_rows($dbc) == 1) {
46         print '<p>The blog entry has been deleted.</p>';
47     } else {
48         print '<p style="color: red;">Could not delete the blog entry because:<br />' .
49         mysqli_error($dbc) . '</p><p>The query being run was: ' . $query . '</p>';
50     }
51 } else { // No ID received.
52     print '<p style="color: red;">This page has been accessed in error.</p>';
53 } // End of main IF.
54
55 mysqli_close($dbc); // Close the connection.
56
57 ?>
58 </body>
59 </html>
```

Switch these
two parameters
around. See red
arrow below.

REMEMBER: We are using mysqli



❷ If the script does not receive an *id* value in the URL, an error is reported.

```
$r = mysqli_query($dbc, $query);
```

3. If the page received a valid entry ID in the URL, define and execute a **SELECT** query:

```
if (isset($_GET['id']) && is_numeric  
→ ($_GET['id'])) {  
    $query = "SELECT title, entry  
→ FROM entries WHERE  
→ entry_id={$_GET['id']}";  
    if ($r = mysqli_query($query,  
→ $dbc)) {
```

To display the blog entry, the page must confirm that a numeric ID is received by the page. Because that value should first come in the URL (when the user clicks the link in **view_blog.php**, see ❷ in the previous section), you reference **\$_GET['id']**.

The query is like the **SELECT** query used in the preceding example, except that the **WHERE** clause has been added to retrieve a specific record. Also, because only the two stored values are necessary—the title and the entry itself—only those are being selected.

This query is then executed using the **mysqli_query()** function.

4. Retrieve the record, and display the entry in a form:

```
$row = mysqli_fetch_array($r);  
print '<form action="delete_entry.  
→ php" method="post">  
<p>Are you sure you want to  
→ delete this entry?</p>  
<p><h3>' . $row['title'] . '</h3>' .  
$row['entry'] . '<br />  
<input type="hidden" name="id"  
→ value="' . $_GET['id'] . '" />  
<input type="submit" name="submit"  
→ value="Delete this Entry!" /></p>  
</form>';
```

continues on next page

REMEMBER: We are using mysqli

Instead of retrieving all the records using a **while** loop, as you did in the previous example, use one call to the **mysql_fetch_array()** function to assign the returned record to the **\$row** variable. Using this array, the record to be deleted can be displayed.

The form first shows the blog entry details, much as it did in the **view_blog.php** script. When the user clicks the button, the form will be submitted back to this page, at which point the record should be deleted. In order to do so, the blog identification number, which is passed to the script as **\$_GET['id']**, must be stored in a hidden input so that it exists in the **\$_POST** array upon submission (because **\$_GET['id']** won't have a value at that point).

5. Report an error if the query failed:

```
} else { // Couldn't get the
→ information.
    print '<p style="color: red;">
→ Could not retrieve the blog
→ entry because:<br />' .
→ mysql_error($dbc) . '</p>
→ <p>The query being run was:
→ ' . $query . '</p>';
}
```

If the **SELECT** query failed to run, the MySQL error and the query itself are printed out.

6. Check for the submission of the form:

```
} elseif (isset($_POST['id']) &&
→ is_numeric($_POST['id']))
→ { // Handle the form.
```

This **elseif** clause is part of the conditional begun in Step 3. It corresponds to the second usage of this same script (the form being submitted). If this conditional is **TRUE**, the record should be deleted.

7. Define and execute the query:

```
$query = "DELETE FROM entries
→ WHERE entry_id={$_POST['id']}
→ LIMIT 1";
$r = mysqli_query($query, $dbc);
```

This query deletes the record whose **entry_id** has a value of **\$_POST['id']**. The ID value comes from the form, where it's stored as a hidden input. By adding the **LIMIT 1** clause to the query, you can guarantee that only one record, at most, is removed.

8. Check the result of the query:

```
if (mysqli_affected_rows($dbc) ==
→ 1) {
    print '<p>The blog entry has
→ been deleted.</p>';
} else {
    print '<p style="color: red;">
→ Could not delete the blog
→ entry because:<br />' .
→ mysql_error($dbc) . '</p>
→ <p>The query being run
→ was: ' . $query . '</p>';
}
```

 **\$r = mysqli_query(\$dbc, \$query);**

The `mysql_affected_rows()` function returns the number of rows altered by the most recent query. If the query ran properly, one row was deleted, so this function should return 1. If so, a message is printed. Otherwise, the MySQL error and query are printed for debugging purposes.

9. Complete the main conditional:

```
} else { // No ID received.  
    print '<p style="color: red;">  
        → This page has been accessed  
        → in error.</p>';  
} // End of main IF.
```

If no numeric ID value was passed to this page using either the GET method or the POST method, then this `else` clause takes effect **C**.

10. Close the database connection, and complete the page:

```
mysql_close($dbc);  
?>  
</body>  
</html>
```

11. Save the script as `delete_entry.php`, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **A** and **B**.

To test this script, you must first run `view_blog.php`. Then, click one of the Delete links to access `delete_entry.php`.

TIP You can empty a table of all of its records by running the query `TRUNCATE TABLE tablename`. This approach is preferred over using `DELETE FROM tablename`. `TRUNCATE` will completely drop and rebuild the table, which is better for the database.

TIP It's a fairly common error to try to run the query `DELETE * FROM tablename`, like a `SELECT` query. Remember that `DELETE` doesn't use the same syntax as `SELECT`, because you aren't deleting specific columns.

Updating Data in a Database

The final type of query this chapter will cover is **UPDATE**. It's used to alter the values of a record's columns. The syntax is

```
UPDATE tablename SET column1_name=  
→ value, column2_name=value2 WHERE  
→ some_column=value
```

As with any other query, if the values are strings, they should be placed within single quotation marks:

```
UPDATE users SET first_name=  
→ 'Eleanor', age=7 WHERE user_id=142
```

As with a **DELETE** query, you should use a **WHERE** clause to limit the rows that are affected. If you don't do this, every record in the database will be updated.

To test that an update worked, you can again use the `mysql_affected_rows()` function to return the number of records altered.

To demonstrate, let's write a page for editing a blog entry. It will let the user alter an entry's title and text, but not the date entered or the blog ID number (as a primary key, the ID number should never be changed). This script will use a structure like that in `delete_entry.php` (Script 12.8), first showing the entry **A**, and then handling the submission of that form **B**.



Browser window: Edit a Blog Entry
URL: phpvqs4:8888/edit_entry.php?id=4


Edit an Entry

Entry Title:

Entry Text:

This is so absolutely amazing that I'm downright speechless! I am still speechless.

A When the user arrives at the edit page, the form is shown with the existing values.



Browser window: Edit a Blog Entry
URL: phpvqs4:8888/edit_entry.php

Edit an Entry

The blog entry has been updated.

B Upon submitting the form, the user sees a message like this.

To update data in a database:

1. Begin a new PHP document in your text editor or IDE, to be named **edit_entry.php (Script 12.9)**.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
→ "http://www.w3.org/TR/xhtml1/
→ DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/
→ 1999/xhtml" xml:lang="en"
→ lang="en">
<head>
    <meta http-equiv="content-type"
→ content="text/html;
→ charset=utf-8" />
    <title>Edit a Blog Entry</title>
</head>
<body>
<h1>Edit an Entry</h1>
```

continues on page 375

Script 12.9 You can edit records in a database table by using an **UPDATE** SQL command.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
4 <head>
5     <meta http-equiv="content-type" content="text/html; charset=utf-8" />
6     <title>Edit a Blog Entry</title>
7 </head>
8 <body>
9 <h1>Edit an Entry</h1>
10 <?php // Script 12.9 - edit_entry.php
11 /* This script edits a blog entry using an UPDATE query. */
12
13 // Connect and select:
14 $dbc = mysql_connect('localhost', 'username', 'password'); require('mysqli_connect.php');
15 mysql_select_db('myblog', $dbc);
16
17 if (isset($_GET['id']) && is_numeric($_GET['id']) ) { // Display the entry in a form:
18
19     // Define the query.
20     $query = "SELECT title, entry FROM entries WHERE entry_id={$_GET['id']}";
21     if ($r = mysql_query($query, $dbc)) { // Run the query. SWITCH
22
23         $row = mysql_fetch_array($r); // Retrieve the information.
24
```

code continues on next page

```
$title = mysqli_real_escape_string($dbc, trim(strip_tags($_POST['title'])));
$entry = mysqli_real_escape_string($dbc, trim(strip_tags($_POST['entry'])));
```

Script 12.9 *continued*

```
25     // Make the form:
26     print '<form action="edit_entry.php" method="post">
27     <p>Entry Title: <input type="text" name="title" size="40" maxsize="100" value="" .
htmlentities($row['title']) . "" /></p>
28     <p>Entry Text: <textarea name="entry" cols="40" rows="5">' . htmlentities($row['entry']) . '</
textarea></p>
29     <input type="hidden" name="id" value="" . $_GET['id'] . "" />
30     <input type="submit" name="submit" value="Update this Entry!" />
31     </form>;
32
33     } else { // Couldn't get the information.
34         print '<p style="color: red;">Could not retrieve the blog entry because:<br />' . mysql_
error($dbc) . '</p><p>The query being run was: ' . $query . '</p>';
35     }
36
37 } elseif (isset($_POST['id']) && is_numeric($_POST['id'])) { // Handle the form.
38
39     // Validate and secure the form data:
40     $problem = FALSE;
41     if (!empty($_POST['title']) && !empty($_POST['entry'])) {
42         $title = mysqli_real_escape_string(trim(strip_tags($_POST['title'])), $dbc);
43         $entry = mysqli_real_escape_string(trim(strip_tags($_POST['entry'])), $dbc);
44     } else {
45         print '<p style="color: red;">Please submit both a title and an entry.</p>;
46         $problem = TRUE;
47     }
48
49     if (!$problem) {
50
51         // Define the query.
52         $query = "UPDATE entries SET title='$title', entry='$entry' WHERE entry_id=
{$_POST['id']}";
53         $r = mysqli_query($query, $dbc); // Execute the query.
54
55         // Report on the result:
56         if (mysqli_affected_rows($dbc) == 1) {
57             print '<p>The blog entry has been updated.</p>;
58         } else {
59             print '<p style="color: red;">Could not update the entry because:<br />' .
mysqli_error($dbc) . '</p><p>The query being run was: ' . $query . '</p>';
60         }
61     } // No problem!
62
63 } else { // No ID set.
64     print '<p style="color: red;">This page has been accessed in error.</p>;
65 } // End of main IF.
66
67
68 mysqli_close($dbc); // Close the connection.
69
70 ?>
71 </body>
72 </html>
```

2. Start your PHP code and connect to the database:

```
<?php // Script 12.9 -
→ edit_entry.php
$dbc = mysql_connect('localhost',
→ 'username', 'password');
mysql_select_db('myblog', $dbc);
```

3. If the page received a valid entry ID in the URL, define and execute a **SELECT** query:

```
if (isset($_GET['id']) &&
→ is_numeric($_GET['id'])) {
    $query = "SELECT title,
→ entry FROM entries WHERE
→ entry_id={$_GET['id']}";
    if ($r = mysql_query($query,
→ $dbc)) {
```

This code is exactly the same as that in the delete page; it selects the two column values from the database for the provided ID value.

4. Retrieve the record, and display the entry in a form:

```
$row = mysql_fetch_array($r);
    print '<form action="edit_entry.
→ php" method="post">
<p>Entry Title: <input type=
→ "text" name="title" size="40"
→ maxsize="100" value="" .
→ htmlentities($row['title']) .
→ "" /></p>
<p>Entry Text: <textarea name=
→ "entry" cols="40" rows="5">' .
→ htmlentities($row['entry']) .
→ '</textarea></p>
<input type="hidden" name="id"
→ value="" . $_GET['id'] . "" />
<input type="submit" name="submit"
→ value="Update this Entry!" />
</form>';
```

Again, this is almost exactly the same as in the preceding script, including the most important step of storing the ID value in a hidden form input. Here, though, the stored data isn't just printed but is actually used as the values for form elements. For security and to avoid potential conflicts, each value is run through **htmlentities()** first.

5. Report an error if the query failed:

```
} else { // Couldn't get the
→ information.
    print '<p style="color:
→ red;">Could not retrieve
→ the blog entry because:
→ <br />' . mysql_error($dbc) .
→ '</p><p>The query being run
→ was: ' . $query . '</p>';
}
```

6. Check for the submission of the form:

```
} elseif (isset($_POST['id']) &&
→ is_numeric($_POST['id'])) {
```

This conditional will be **TRUE** when the form is submitted.

7. Validate and secure the form data:

```
$problem = FALSE;
if (!empty($_POST['title']) &&
→ !empty($_POST['entry'])) {
    $title = mysql_real_escape_
→ string(trim(strip_tags($_POST
→ ['title'])), $dbc);
    $entry = mysql_real_escape_
→ string(trim(strip_tags($_POST
→ ['entry'])), $dbc);
} else {
    print '<p style="color: red;">
→ Please submit both a title
→ and an entry.</p>';
    $problem = TRUE;
}
```

continues on next page

This code comes from the page used to add blog postings. It performs minimal validation on the submitted data and then runs it through the `mysql_real_escape_string()` function to be safe. Because the form data can be edited, the form should be validated as if it were a new record being created.

8. Define and execute the query:

```
if (!$problem) {
    $query = "UPDATE entries SET
        → title='$title', entry='$entry'
        → WHERE entry_id={$_POST['id']}";
    $r = mysql_query($query, $dbc);
```

The **UPDATE** query sets the **title** column equal to the value entered in the form's title input and the **entry** column equal to the value entered in the form's entry text area. Only the record whose **entry_id** is equal to `$_POST['id']`, which comes from a hidden form input, is updated.

9. Report on the success of the query:

```
if (mysql_affected_rows($dbc) == 1) {
    print '<p>The blog entry has
        → been updated.</p>';
} else {
    print '<p style="color: red;">
        → Could not update the entry
        → because:<br />' . mysql_
        → error($dbc) . '</p><p>The
        → query being run was: ' .
        → $query . '</p>';
}
```

If one row was affected, then a success message is returned. Otherwise, the MySQL error and the query are sent to the Web browser.

10. Complete the conditionals:

```
    } // No problem!
} else { // No ID set.
    print '<p style="color: red;">
        → This page has been accessed
        → in error.</p>';
} // End of main IF.
```

If no numeric ID value was passed to this page using either the GET method or the POST method, then this **else** clause takes effect.

11. Close the database connection, and complete the page:

```
mysql_close($dbc);
?>
</body>
</html>
```

12. Save the file as **edit_entry.php**, place it in the proper directory for your PHP-enabled server, and test it in your Web browser **A** and **B**.

As in the preceding example, to edit an entry, you must click its *Edit* link in the **view_blog.php** page.

13. Revisit `view_blog.php` to confirm that the changes were made **C**.

TIP The id is a primary key, meaning that its value should never change. By using a primary key in your table, you can change every other value in the record but still refer to the row using that column.

TIP The `mysql_real_escape_string()` function does not need to be applied to the ID values used in the queries, as the `is_numeric()` test confirms they don't contain apostrophes or other problematic characters.

TIP More thorough edit and delete pages would use the `mysql_num_rows()` function in a conditional to confirm that the SELECT query returned a row prior to fetching it:

```
if (mysql_num_rows($r) == 1) {...
```

TIP If you run an update on a table but don't change a record's values, `mysql_affected_rows()` will return 0.

TIP It can't hurt to add a `LIMIT 1` clause to an UPDATE query, to ensure that only one row, at most, is affected.



C Reloading the `view_blog.php` script reflects the changes made to the entries.