CHAPTER

9

MANIPULATING MySQL DATABASES WITH PHP

In this chapter, you will:

- Connect to MySQL from PHP
- Learn how to handle MySQL errors
- Execute SQL statements with PHP
- Use PHP to work with MySQL databases and tables
- Use PHP to manipulate database records

One of PHP's greatest strengths is its ability to access and manipulate databases. With its strong ODBC support, you can use PHP to access any database that is ODBC compliant. PHP also includes functionality that allows you to work directly with different types of databases, without going through ODBC. Some of the databases that you can access directly from PHP include Oracle, Informix, PostgresSQL, and MySQL.

PHP also supports other methods of accessing data sources, including SQLite, database abstraction layer functions, and PEAR DB. SQLite and database abstraction layer functions work with file-based databases instead of server-based databases such as MySQL. The PHP Extension and Application Repository (PEAR) is a library of open source PHP code. One of the most popular PEAR code modules is PEAR DB, which simplifies access between PHP and a database server by providing a generic interface that works with various types of database systems, similar to the way ODBC works. Although PEAR DB and ODBC perform similar functions, the difference between the two languages is that PEAR is designed specifically to work with PHP, whereas ODBC is a more generic protocol that is used by many programming languages and database management systems.

With so many database connectivity options, how do you decide which method to use for accessing databases with PHP? First, you need to select a database management system. If you are new to database development, you should probably start with an open source database such as PostgresSQL or

MySQL, mainly because they are free and fairly easy to learn. After you select a database, you need to determine whether PHP can access it directly or whether it must go through a layer such as ODBC or PEAR DB. Going through ODBC or PEAR DB makes it easier for you to write PHP code that can be used with a variety of databases. However, your PHP script will be faster if it can access a database directly, without going through a PEAR DB or ODBC layer. Therefore, if you anticipate that your PHP script will need to access more than one type of database, you should use PEAR DB or ODBC. To be more precise, you should use PEAR DB over ODBC because PEAR is designed specifically for the PHP language. Yet, there are cases when ODBC is preferable, especially when you need to access Microsoft data source products such as Microsoft Access or Microsoft Excel. However, if you plan to work with a single database, such as MySQL, and you are more concerned with your Web application's performance than whether it is compatible with multiple database systems, use PHP's direct database access functionality if it's available for your database management system.

In this chapter, you study how to use PHP to directly access MySQL.

CONNECTING TO MYSQL WITH PHP

As you work through this chapter, keep in mind that almost everything you learned in the preceding chapter about MySQL is applicable to this chapter. Although you need to learn a few new functions to access MySQL with PHP, you will execute the same SQL statements that you used with the MySQL Monitor. The great benefit to using PHP or some other server-side scripting language to read from and write to a database server is that it allows you to create a Web-based interface that makes it much easier for visitors to interact with your database.

Before you can use PHP to read from and write to MySQL databases, you need to enable MySQL support in PHP and learn how to connect to the MySQL database server.

Enabling MySQL Support in PHP

In PHP versions earlier than PHP 5, support for MySQL was installed by default. However, starting with PHP 5, MySQL support no longer comes preinstalled with PHP. To enable MySQL support in PHP, you must configure your PHP installation to use the `mysqli` extension.

NOTE

The `mysqli` extension is designed to work with MySQL version 4.1.3 and higher. If you are using a version of MySQL that is older that 4.1.3, you must use the `mysql` extension.

How you enable MySQL support in PHP depends on your operating system and how you installed PHP. On UNIX/Linux systems, you configure PHP to use the `mysqli` extension by specifying the `--with-mysqli` parameter when you run the `configure` command during the PHP installation process. If you followed the UNIX/Linux PHP installation instructions in Chapter 2, you should have specified the `--with-mysqli` parameter when you ran the `configure` command. If you did not specify the `--with-mysqli` parameter when you ran the `configure` command, you need to reinstall PHP to complete the exercises in this chapter.

To enable the `mysqli` extension on Windows installations of PHP, you must copy two files, libmysql.dll and php_mysqli.dll, to the directory where you installed PHP. You must also edit your php.ini configuration file and enable the `extension=php_mysqli.dll` directive. The libmysql.dll and php_mysqli.dll files are available in the full PHP Windows zip package (not the Windows binary installer) that is available on the PHP download page. The following instructions describe how to enable MySQL support on Windows installations of PHP.

TIP

If you are working with an installation of PHP that is hosted by an ISP, MySQL support should already be enabled.

NOTE

The following steps assume that you followed the instructions in Chapter 2 to install PHP with the Windows binary installer. If you installed PHP by using the full PHP Windows zip package, the libmysql.dll file is installed by default in the PHP installation directory and the php_mysqli.dll file is installed in the ext directory beneath the PHP installation directory. You only need to copy the php_mysqli.dll file from the ext directory to the main PHP installation directory and enable the `extension=php_mysqli.dll` directive, as described in the following steps.

To enable MySQL support on Windows installations of PHP:

1. Start your Web browser, and enter the Web address for the PHP download page: **http://www.php.net/downloads.php**. Download the Windows zip package (not the Windows binary installer) containing the most recent Windows binary files. Save the file to a temporary folder on your computer.
2. Open **Windows Explorer** or **My Computer** and navigate to the folder where you downloaded the Windows zip package. Double-click the file to open it in WinZip, which is the archive utility for Windows.
3. Extract the **libmysql.dll** and **php_mysqli.dll** files to the directory where you installed PHP. By default, the PHP installation directory is C:\PHP. To extract individual files in WinZip, locate and click on a filename, and then click the **Extract** button. The Extract dialog box opens, which allows you to specify the location where you want to store the file.
4. Close **WinZip**.

5. Open your **php.ini** configuration file in your text editor. On Windows systems, this file is installed automatically in your main Windows directory, which is usually C:\WINDOWS or C:\WINNT.
6. In the php.ini file, locate the **`extension=php_mysqli.dll`** directive (not the `extension=php_mysql.dll` directive) and remove the semicolon at the beginning of the line to enable MySQL support. If the `extension=php_mysqli.dll` directive does not exist in your **php.ini** file, add it to the end of the Windows Extensions section.
7. Save and close the **php.ini** file.
8. Restart your Web server.

TIP

See Chapter 2 for information on how to restart your Web server.

Opening and Closing a MySQL Connection

Before you can use PHP to access the records in a database, you must first use the `mysqli_connect()` function to open a connection to a MySQL database server. Opening a connection to a database is similar to opening a handle to a text file, as you did in Chapter 6. However, instead of returning a file handle, the `mysqli_connect()` function returns a positive integer if it connects to the database successfully or false if it doesn't. You assign the return value from the `mysqli_connect()` function to a variable that you can use to access the database in your script. The basic syntax for the `mysqli_connect()` function is as follows:

```
$connection = mysqli_connect("host"[, "user ", "password", "database"])
```

In the preceding, the *host* argument allows you to specify the host name where your MySQL database server is installed. If you are working with an instance of MySQL database server that is installed on your local computer, use a value of "localhost" or "127.0.0.1" for the *host* argument. However, if you are working with a MySQL database server on an ISP's Web site, you need to enter your ISP's host name. The *user* and *password* arguments allow you to specify a MySQL account name and password, and the *database* argument allows you to select a database with which to work. For example, the following command connects the username *dongosselin* with a password of "rosebud" to a local instance of MySQL database server and opens a database named `real_estate`. The database connection is assigned to the `$DBConnect` variable.

```
$DBConnect = mysqli_connect("localhost", "dongosselin",
     "rosebud", "real_estate");
```

TIP

To change users after connecting to a database, use the `mysqli_change_user()` function.

When your PHP script ends, any open database connections close automatically. However, you should get into the habit of explicitly closing database connections with the **`mysqli_close()`** function when you are finished with them to ensure that the connection doesn't keep taking up space in your computer's memory while the script finishes processing. You close a database connection by passing the database connection variable to the **`mysqli_close()`** function. The following statement closes the `$DBConnect` database connection variable that was opened in the preceding statement:

```
mysqli_close($DBConnect);
```

TIP

If you receive a warning that PHP is unable to load a dynamic library or an error such as "Call to undefined function `mysqli_connect()`," MySQL support is not correctly enabled for your PHP installation. For more information, refer to the "Enabling MySQL Support in PHP" section earlier in this chapter.

After you connect to a database with the **`mysqli_connect()`** function, you can use the functions listed in Table 9-1 to return information about your installation of MySQL server.

Table 9-1 MySQL server information functions

Function	Description
`mysqli_get_client_info()`	Returns the MySQL client version
`mysqli_get_client_version()`	Returns the MySQL client version as an integer
`mysqli_get_host_info(`*`connection`*`)`	Returns the MySQL database server connection information
`mysqli_get_proto_info(`*`connection`*`)`	Returns the MySQL protocol version
`mysqli_get_server_info(`*`connection`*`)`	Returns the MySQL database server version
`mysqli_get_server_version(`*`connection`*`)`	Returns the MySQL database server version as an integer

NOTE

The `mysqli_get_client_info()` and `mysqli_get_client_version()` functions do not accept any arguments. However, you must pass the variable representing the database connection to the rest of the functions listed in Table 9-1.

Next, you create a PHP script that connects to MySQL and uses the functions listed in Table 9-1 to print information about your installation of MySQL.

To create a PHP script that connects to MySQL and uses the functions listed in Table 9-1 to print information about your installation of MySQL:

1. Create a new document in your text editor.
2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and `<body>` element. Use the strict DTD and "MySQL Server Information" as the content of the `<title>` element.
3. Add the following `<link>` element above the closing `</head>` tag to link to the php_styles.css style sheet in your Chapter directory:

   ```
   <link rel="stylesheet" href="php_styles.css" type="text/css" />
   ```
4. Add the following heading element to the document body:

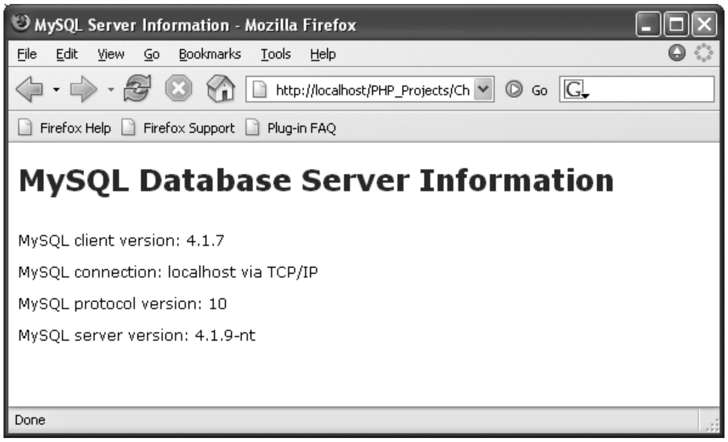
   ```
   <h1>MySQL Database Server Information</h1>
   ```
5. Add the following script section to the end of the document body:

   ```
   <?php
   ?>
   ```
6. Add the following `mysqli_connect()` statement to the script section. Replace *`user`* and *`password`* with the MySQL username and password you created in Chapter 8.

   ```
   $DBConnect = mysqli_connect("localhost", "user", "password");
   ```
7. Add to the end of the script section the following statements, which print information about your installation of MySQL server:

   ```
   echo "<p>MySQL client version: "
        . mysqli_get_client_info() . "</p>";
   echo "<p>MySQL connection: "
        . mysqli_get_host_info($DBConnect) . "</p>";
   echo "<p>MySQL protocol version: "
        . mysqli_get_proto_info($DBConnect) . "</p>";
   echo "<p>MySQL server version: "
        . mysqli_get_server_info($DBConnect) . "</p>";
   ```
8. Finally, add the following statement to the end of the script section to close the database connection:

   ```
   mysqli_close($DBConnect);
   ```
9. Save the document as **MySQLInfo.php** in the Chapter directory for Chapter 9, and then close it in your text editor.
10. Open the **MySQLInfo.php** file in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/MySQLInfo.php**. Your Web browser should appear similar to Figure 9-1, although the information printed from each function might be different for your MySQL installation.


MySQL Server Information - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Ch
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

MySQL Database Server Information

MySQL client version: 4.1.7

MySQL connection: localhost via TCP/IP

MySQL protocol version: 10

MySQL server version: 4.1.9-nt

Figure 9-1 MySQLInfo.php in a Web browser

11. Close your Web browser window.

Selecting a Database

As you saw in Chapter ~~6~~, you must first select a database with the `use` ***`database`*** statement when you log on to the MySQL Monitor. Although you can select a database by passing a database name as the fourth argument to the `mysqli_connect()` function, you can also select or change a database with the `mysqli_select_db()` function. The syntax for the `mysqli_select_db()` function is `mysqli_select_db(`*`connection`*`,` *`database`*`)`. The function returns a value of true if it successfully selects a database or false if it doesn't. For example, instead of selecting a database by passing the database name as the fourth argument to the `mysqli_connect()` function, the following code uses a `mysqli_select_db()` statement to open the `real_estate` database from the `$DBConnect` database connection:

```
$DBConnect = mysqli_connect("localhost", "dongosselin", "rosebud");
mysqli_select_db($DBConnect, "real_estate");
// additional statements that access or manipulate the database
mysqli_close($DBConnect);
```

Next, you create a PHP script that selects the `flightlog` database you created in Chapter 8.

To create a PHP script that selects the `flightlog` database you created in Chapter 8:

1. Create a new document in your text editor.

2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and `<body>` element. Use the strict DTD and "Flightlog Entries" as the content of the `<title>` element.

3. Add the following `<link>` element above the closing `</head>` tag to link to the php_styles.css style sheet in your Chapter directory:

```
<link rel="stylesheet" href="php_styles.css" type="text/css" />
```

4. Add the following heading element to the document body:

```
<h1>Flightlog Entries</h1>
```

5. Add the following script section to the end of the document body:

```
<?php
?>
```

6. Add the following `mysqli_connect()` statement to the script section. Replace *user* and *password* with the MySQL username and password you created in Chapter 8.

```
$DBConnect = mysqli_connect("localhost", "user", "password");
```

7. After the `mysqli_connect()` function, add the following statements to select the `flightlog` database:

```
$DBName = "flightlog";
mysqli_select_db($DBConnect, $DBName);
```

8. Add the following statement to the end of the script section to close the database connection:

```
mysqli_close($DBConnect);
```

9. Save the document as **FlightlogEntries.php** in the Chapter directory for Chapter 9, and then close it in your text editor.

Handling MySQL Errors

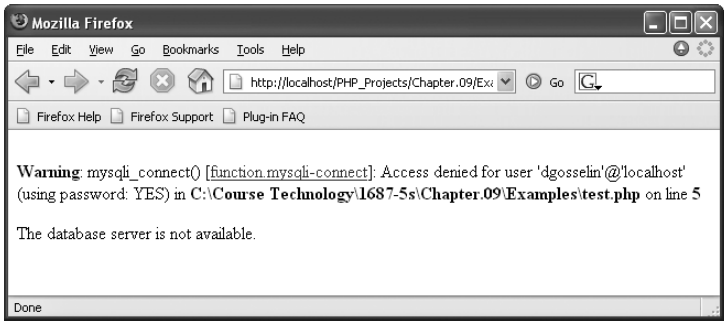
When accessing MySQL databases and other types of data sources, you need to understand the errors that can affect the execution of your script. One of the most important errors that you need to consider occurs when you cannot connect to a database server. Reasons that you may not be able to connect to a database server include the following:

- The database server is not running.
- You do not have sufficient privileges to access the data source.
- You entered an invalid username and/or password.

When it comes to connecting to a database server or selecting a database, your first instinct might be to check the value that is returned from the `mysqli_connect()` function and `mysqli_select_db()` function. The `mysqli_connect()` function returns a positive integer if it connects to the database successfully or false if it doesn't. Thus, you might make the mistake of trying to use the return value to determine if the connection is successful. As an example, consider the following code. The conditional expression in the `if` statement uses the Not operator (`!`) to determine if the `$DBConnect` variable is equal to false. If the variable is equal to false, a message prints to the Web browser informing the user that the database server is not available. If the variable is not equal to false, a message prints to the Web browser informing the user that she connected successfully to the database server.

```
$DBConnect = mysqli_connect("localhost", "dongosselin",
     "rosebud", "flightlog");
if (!$DBConnect)
      echo "<p>The database server is not available.</p>";
else   {
     echo "<p>Successfully connected to the database server.</p>";
     // additional statements that access the database server
     mysqli_close($DBConnect);
}
```

The problem with the preceding code is that, although it prints "The database server is not available" if you cannot connect to the database server, it also prints any error messages that may be caused by the `mysqli_connect()` function. For example, Figure 9-2 displays an error message that occurs if you attempt to access the database with an invalid username or password.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapter.09/Ex
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

Warning: mysqli_connect() [function.mysqli-connect]: Access denied for user 'dgosselin'@'localhost' (using password: YES) in **C:\Course Technology\1687-5s\Chapter.09\Examples\test.php** on line **5**

The database server is not available.

Figure 9-2 Database connection error message

As with the connection to the database server, you should also check to ensure that the `mysqli_select_db()` function successfully selects the database. Because the `mysqli_select_db()` function returns a value of true if it is successful, you can call the function from within an `if` statement's conditional expression. The following code demonstrates how to call the `mysqli_select_db()` function from within an `if` statement's conditional expression to determine whether the database was selected successfully:

```
$DBConnect = mysqli_connect("localhost", "dongosselin", "rosebud");
if (!$DBConnect)
      echo "<p>The database server is not available.</p>";
else   {
```

```
         echo "<p>Successfully connected to the database server.</p>";
          if (mysqli_select_db($DBConnect, "flightlog")) {
                echo "<p>Successfully opened the database.</p>";
                // additional statements that access the database
          }
          else
                echo "<p>The database is not available.</p>";
          mysqli_close($DBConnect);
     }
```

In most cases, the `mysqli_select_db()` function does not print any error messages the way the `mysqli_connect()` function does. However, to be on the safe side, you should suppress any error codes that may appear for both the `mysqli_connect()` function and the `mysqli_select_db()` function. In the next section, you learn how to suppress errors with the error control operator.

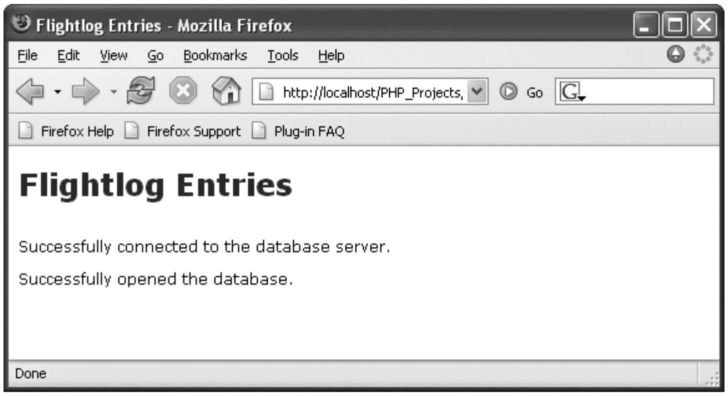
Next, you modify the FlightlogEntries.php script so it verifies that the database is connected and that the `flightlog` database is selected.

To modify the FlightlogEntries.php script so it verifies that the database is connected and that the `flightlog` database is selected:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Modify the script section so it verifies that the database is connected and that the `flightlog` database is selected, as follows:

```
$DBConnect = mysqli_connect("localhost", "user", "password");
if (!$DBConnect)
     echo "<p>The database server is not available.</p>";
else  {
echo "<p>Successfully connected to the database server.</p>";
     $DBName = "flightlog";
     if (mysqli_select_db($DBConnect, $DBName))
          echo "<p>Successfully opened the database.</p>";
     else
          echo "<p>The database is not available.</p>";
     mysqli_close($DBConnect);
}
```

3. Save the **FlightlogEntries.php** file and open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. Your Web browser should appear similar to Figure 9-3.


Flightlog Entries - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects,
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

Flightlog Entries

Successfully connected to the database server.

Successfully opened the database.

Figure 9-3 FlightlogEntries.php in a Web browser

4. Close your Web browser window.

Suppressing Errors with the Error Control Operator

9

Although standard error messages that are generated by programming languages such as PHP are very helpful to programmers, they tend to scare users, who might think that they somehow caused the error. Errors can and will occur, but you should never let your users think that they did something wrong. Your goal should be to write code that anticipates any problems that may occur and includes graceful methods of dealing with those problems. Writing code that anticipates and handles potential problems is often called **bulletproofing**. One bulletproofing technique you have already used has to do with validating submitted form data. For example, in Chapter 5, you saw the following code, which ensures that values submitted to a script that measures body mass contain numeric values. This example contains a nested `if` statement that tests whether the `$_GET['height']` and `$_GET['weight']` variables are numeric after the first `if` statement checks to see whether they are set.

```
if (isset($_GET['height']) && isset($_GET['weight'])) {
     if (!is_numeric($_GET['weight']) || !is_numeric($_GET['height'])) {
          $BodyMass = $_GET['weight'] / ($_GET['height']
                * $_GET['height']) * 703;
          printf("<p>Your body mass index is %d.</p>",
                $BodyMass);
     }
     else
          echo "<p>You must enter numeric values!</p>";
}
```

Another method of bulletproofing your code is to use the **error control operator (@)** to suppress error messages. You can place the error control operator before any expression, although it is most commonly used with built-in PHP functions, especially functions that access data sources such as the `mysqli_connect()` and `mysqli_select_db()` functions. Using the error control operator to suppress error messages does not mean you can then ignore errors that may occur. Instead, the error control operator allows you to provide a more graceful way of handling an error instead of allowing an intimidating error message to be printed to the Web browser. The following example contains a modified version of the code that connects with a username and password of "dongosselin" and "rosebud" to the `flightlog` database. In this example, both the `mysqli_connect()` and `mysqli_select_db()` functions are preceded by error control operators to suppress any error messages that may occur.

```
$DBConnect = @mysqli_connect("localhost", "dongosselin", "rosebud");
if (!$DBConnect)
      echo "<p>The database server is not available.</p>";
else   {
      echo "<p>Successfully connected to the database server.</p>";
      if (@mysqli_select_db($DBConnect, "flightlog")) {
            echo "<p>Successfully opened the database.</p>";
      // additional statements that access the database
     }
     else
           echo "<p>The database is not available.</p>";
     mysqli_close($DBConnect);
}
```

Next, you add error control operators to the `mysqli_connect()` and `mysqli_select_db()` functions in the FlightlogEntries.php script.

To add error control operators to the `mysqli_connect()` and `mysqli_select_db()` functions in the FlightlogEntries.php script:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Add error control operators before the `mysqli_connect()` and `mysqli_select_db()` functions.
3. Save the **FlightlogEntries.php** file and open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. The Web page should appear the same as it did before you added the error control operators.
4. Close your Web browser window.

Terminating Script Execution

Up to this point in this book, you have relied on `if...else` statements to execute code only when certain conditions have been met. For example, the `else` clause in the preceding example executes only if the `mysqli_connect()` successfully connects to a database server. The `else` clause also contains a nested `if` statement, which executes only if the `mysqli_select_db()` function successfully opens a database. Instead of relying on `if...else` statements to execute code, you can more easily terminate script execution with the `die()` or `exit()` functions. The `die()` and `exit()` functions perform the same task of terminating script execution, although the `die()` version is usually used when attempting to access a data source. Both functions accept a single string argument, which is printed to the Web browser when the script ends. You can call the `die()` and `exit()` functions as separate statements or by appending either function to an expression with the `Or` operator. The following code demonstrates how to call the `die()` function by using `if` statements. Notice that the code does not require `else` clauses because the script terminates when the conditional expressions in the `if` statements are true.

```
$DBConnect = @mysqli_connect("localhost", "root", "paris");
if (!$DBConnect)
     die("<p>The database server is not available.</p>");
echo "<p>Successfully connected to the database server.</p>";
$DBSelect = @mysqli_select_db($DBConnect, "flightlog");
if (!$DBSelect)
     die("<p>The database is not available.</p>");
echo "<p>Successfully opened the database.</p>";
// additional statements that access the database
mysqli_close($DBConnect);
```

The following code demonstrates how to use an Or operator to append the `die()` function to the statements that call the `mysqli_connect()` and `mysqli_select_db()` functions:

```
$DBConnect = @mysqli_connect("localhost", "dongosselin",
"rosebud")
     Or die("<p>The database server is not available.</p>");
echo "<p>Successfully connected to the database server.</p>";
@mysqli_select_db($DBConnect, "flightlog")
     Or die("<p>The database is not available.</p>");
echo "<p>Successfully opened the database.</p>";
// additional statements that access the database server
mysqli_close($DBConnect);
```

Next, you modify the `mysqli_connect()` and `mysqli_select_db()` functions in the FlightlogEntries.php script so they use `die()` functions to terminate the script in the event of an error.

To modify the `mysqli_connect()` and `mysqli_select_db()` functions in the FlightlogEntries.php script so they use `die()` functions to terminate the script in the event of an error:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Modify the contents of the script section so it uses `die()` functions to terminate the script in the event of an error. Your modified script section should appear as follows:

```
$DBConnect = @mysqli_connect("localhost", "user", "password")
     Or die("<p>The database server is not available.</p>");
echo "<p>Successfully connected to the database server.</p>";
$DBName = "flightlog";
@mysqli_select_db($DBConnect, $DBName)
     Or die("<p>The database is not available.</p>");
echo "<p>Successfully opened the database.</p>";
mysqli_close($DBConnect);
```

3. Save the **FlightlogEntries.php** file and open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. The Web page should appear the same as it did before you added the error control operators.
4. Close your Web browser window.

Reporting MySQL Errors

The preceding section emphasized the importance of using the error control operator to prevent PHP from spitting out errors wherever they occur. However, that does not mean error messages are useless. In fact, when displayed correctly, error numbers and codes can be invaluable in providing useful feedback and in helping you track down problems with your script or database. PHP includes the functions listed in Table 9-2 for reporting MySQL error numbers and codes.

Table 9-2 MySQL error reporting functions

Function	Description
`mysqli_connect_errno()`	Returns the error code from the last database connection attempt or zero if no error occurred
`mysqli_connect_error()`	Returns the error message from the last database connection attempt or an empty string if no error occurred
`mysqli_errno(`*`connection`*`)`	Returns the error code from the last attempted MySQL function call or zero if no error occurred
`mysqli_error(`*`connection`*`)`	Returns the error message from the last attempted MySQL function call or an empty string if no error occurred
`mysqli_sqlstate(`*`connection`*`)`	Returns a string of five characters representing an error code from the last MySQL operation or 00000 if no error occurred

TIP

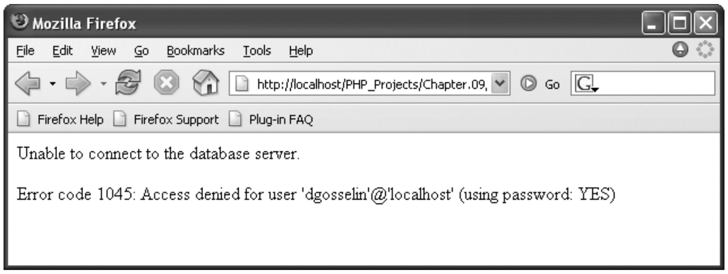
You can find a list of error codes that may be returned from the `mysqli_sqlstate()` function at *http://dev.mysql.com/doc/mysql/en/error-handling.html*.

As an example of how you might use a MySQL error reporting function, consider a PHP script that allows users to submit a username and password that will be used to log on to MySQL. For example, a Web page may contain the following simple form that will be submitted to a PHP script named dblogin.php:

```
<form action="dblogin.php" method="GET"
enctype="application/x-www-form-urlencoded">
<p>Username <input type="text" name="username" /><br />
Password <input type="password" name="password" /></p>
<p><input type="submit" value="Log In" /></p>
</form>
```

If a user enters an invalid username or password with the preceding form, printing a generic message such as "The database server is not available" doesn't help him determine what's wrong. When connecting to the MySQL database server, you should at least use the `mysqli_connect_error()` function to give the user more information about the error that occurred. For example, the `die()` function in the following code uses the `mysqli_connect_errno()` and the `mysqli_connect_error()` functions to print an error code and message if the connection attempt fails. Both of these functions report on the most recent database connection attempt. If the user enters an invalid username or password in the form, he will see the error number and description shown in Figure 9-4. As you can see in the figure, the error description informs users that access was denied for the submitted username and password, which should point out that either a typo occurred in the submitted username or password, or that the user doesn't have authorization to access the database.

```
$User = $_GET['username'];
$Password = $_GET['password'];
$DBConnect = @mysqli_connect("localhost", $User, $Password)
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
echo "<p>Successfully connected to the database server.</p>";
@mysqli_select_db($DBConnect, "flightlog")
     Or die("<p>The database is not available.</p>");
echo "<p>Successfully opened the database.</p>";
// additional statements that access the database
mysqli_close($DBConnect);
```


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapter.09,
Go
Firefox Help Firefox Support Plug-in FAQ

Unable to connect to the database server.

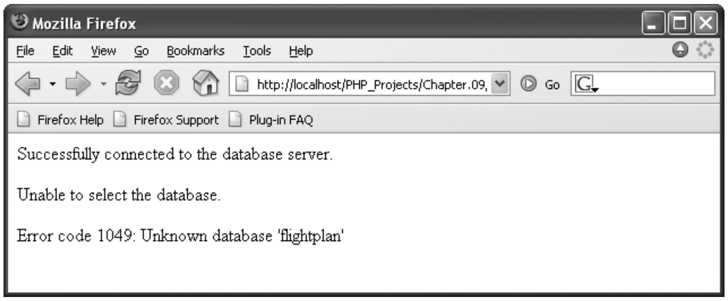
Error code 1045: Access denied for user 'dgosselin'@'localhost' (using password: YES)

Figure 9-4 Error number and message generated by an invalid username and password

The `mysqli_connect_errno()` and `mysqli_connect_error()` functions only report errors that occur when you attempt to connect to a MySQL database server with the `mysqli_connect()`function. To obtain error information for any other functions that access a MySQL database, such as the `mysqli_select_db()` function, you use the `mysqli_errno()` and the `mysqli_error()` functions. Unlike the `mysqli_connect_errno()` and the `mysqli_connect_error()` functions, you pass to the the `mysqli_errno()` and the `mysqli_error()` functions the variable representing the database connection. The following example demonstrates how to display error codes and messages that may occur when you call the `mysqli_select_db()` function:

```
$User = $_GET['username'];
$Password = $_GET['password'];
$DBConnect = @mysqli_connect("localhost", $User, $Password)
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
echo "<p>Successfully connected to the database server.</p>";
@mysqli_select_db($DBConnect, "flightplan")
     Or die("<p>Unable to select the database.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully opened the database.</p>";
// additional statements that access the database
mysqli_close($DBConnect);
```

The preceding script attempts to select a database named `flightplan`. Figure 9-5 shows the output in a Web browser if the `flightplan` database does not exist on the MySQL database server.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapter.09,
Go
Firefox Help
Firefox Support
Plug-in FAQ

```
Successfully connected to the database server.

Unable to select the database.

Error code 1049: Unknown database 'flightplan'
```

Figure 9-5 Error code and message generated when attempting to select a database that does not exist

Next, you modify the `die()` functions in the FlightlogEntries.php script so they print error codes and messages in the event of an error.

To modify the `die()` functions in the FlightlogEntries.php script so they print error codes and messages in the event of an error:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Modify the `die()` function in the `mysqli_connect()` statement so it includes the `mysqli_connect_errno()` and `mysqli_connect_error()` functions, as follows:

```
$DBConnect = @mysqli_connect("localhost",
"dongosselin","rosebud")
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
```

3. Modify the `die()` function in the `mysqli_select_db()` statement so it includes the `mysqli_errno()` and `mysqli_error()` functions, as follows:

```
@mysqli_select_db($DBConnect, $DBName)
     Or die("<p>Unable to select the database.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
```

4. Save the **FlightlogEntries.php** file and open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. The Web page should appear the same as it did before you added the error functions.
5. Close your Web browser window.

Executing SQL Statements

In this section, you learn how to use PHP to submit SQL statements to MySQL. As you work through the rest of this chapter, you should recognize the SQL statements because you worked with all of them in Chapter 8. The primary difference is that instead of manually executing SQL statements by typing them in the MySQL Monitor as you did in Chapter 8, you use PHP statements to access MySQL and execute SQL statements for you.

In PHP, you use the `mysqli_query()` function to send SQL statements to MySQL. The `mysqli_query()` function is the workhorse of PHP connectivity with MySQL; almost every SQL command you send to MySQL from PHP is executed with the `mysqli_query()` function. The basic syntax for the `mysqli_query()` function is `mysqli_query(`*`connection`*`, `*`query`*`)`. The `mysqli_query()` function returns one of three values, depending on the type of query executed. For SQL statements that do not return results, such as the `CREATE DATABASE` and `CREATE TABLE` statements, the `mysqli_query()` function returns a value of true if the statement executes successfully. For SQL statements that return results, such as `SELECT` and `SHOW` statements, the `mysqli_query()` function returns a result pointer that represents the query results. A **result pointer** is a special type of variable that refers to the currently selected row in a resultset. The query pointer is a way of keeping track of where you are in a resultset. You assign the result pointer to a variable, which you can use to access the resultset in PHP. The `mysqli_query()` function returns a value of false for any SQL statements that fail, regardless of whether they return results. As an example, the following code queries the `guitars` database you saw in Chapter 8. The code then executes the `mysqli_query()` function and assigns the result pointer to a variable named `$QueryResult`.

```
@mysqli_select_db($DBConnect, "guitars")
     Or die("<p>Unable to select the database.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully opened the database.</p>";
$SQLstring = "SELECT model, quantity FROM inventory";
$QueryResult = mysqli_query($DBConnect, $SQLstring)
mysqli_close($DBConnect);
```

You use the same techniques to handle errors with the `mysqli_query()` function that you use with the `mysqli_connect()` and `mysqli_select_db()` functions. For example, the following code uses the error control operator to suppress errors and terminates the script with the `die()` function if the query is unsuccessful. The example also uses the `mysqli_errno()` and `mysqli_error()` functions to report the error code and message.

```
$SQLstring = "SELECT model, quantity FROM inventory";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
```

```
        Or die("<p>Unable to execute the query.</p>"
        . "<p>Error code " . mysqli_errno($DBConnect)
        . ": " . mysqli_error($DBConnect)) . "</p>";
    echo "<p>Successfully executed the query.</p>";
    mysqli_close($DBConnect);
```

When you use a PHP variable to represent a field name in a SQL query, you must enclose the variable name within single quotes or you receive an error. For example, the following statement raises an error because the `$Make` variable is not enclosed within single quotes:

```
$Make = "Ovation";
$SQLstring = "SELECT model, quantity FROM $DBTable
     WHERE model=$Make";
```

To fix the preceding code, enclose the `$Make` variable in single quotes, as follows:

```
$Make = "Ovation";
$SQLstring = "SELECT model, quantity FROM $DBTable
     WHERE model='$Make'";
```

Next, you add query statements to the FlightlogEntries.php script that select all the records in the `flightsessions` table.

To query statements to the FlightlogEntries.php script that select all the records in the `flightsessions` table:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Add the following statements above the `mysqli_close()` statement. The first statement creates a SQL query that selects all records from the `flightsessions` table. The second statement executes the query with the `mysqli_query()` function, and the third statement prints a message if the query is successful.

```
$SQLstring = "SELECT * FROM flightsessions";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully executed the query.</p>";
```

3. Save the **FlightlogEntries.php** file and open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. You should see the three success messages printed to the Web browser window.
4. Close your Web browser window.

Working with Query Results

Recall that for SQL statements that return results, such as `SELECT` and `SHOW` statements, the `mysqli_query()` function returns a result pointer that represents the query results. You assign the result pointer to a variable, which you can use to access the resultset in PHP. To access the database records through the result pointer, you must use one of the functions listed in Table 9-3.

Table 9-3 Common PHP functions for accessing database results

Function	Description
`mysqli_data_seek(`*`$Result, position`*`)`	Moves the result pointer to a specified row in the resultset
`mysqli_fetch_array(`*`$Result`*`, MYSQLI_ASSOC | MYSQLI_NUM | MYSQLI_BOTH)`	Returns the fields in the current row of a resultset into an indexed array, associative array, or both and moves the result pointer to the next row
`mysqli_fetch_assoc(`*`$Result`*`)`	Returns the fields in the current row of a resultset into an associative array and moves the result pointer to the next row
`mysqli_fetch_lengths(`*`$Result`*`)`	Returns the field lengths for the current row in a resultset into an indexed array
`mysqli_fetch_row(`*`$Result`*`)`	Returns the fields in the current row of a resultset into an indexed array and moves the result pointer to the next row

First, you learn how to use the `mysqli_fetch_row()` function to retrieve fields into an indexed array.

Retrieving Records into an Indexed Array

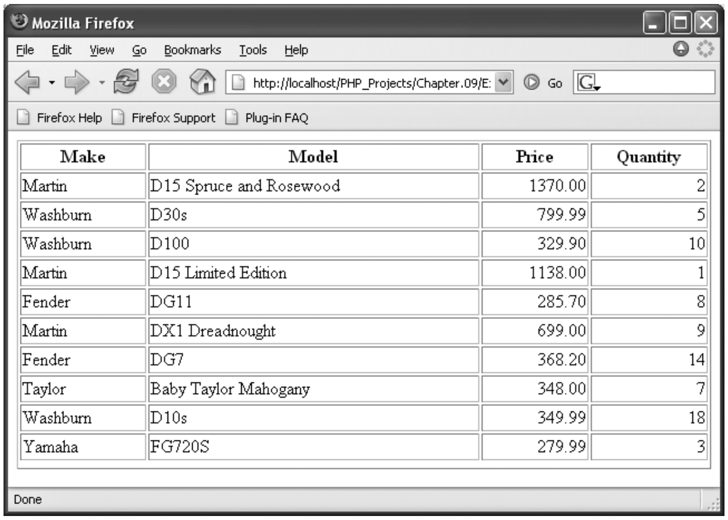
In Chapter 6, you learned how to use the `fgets()` function, which returns a line from a text file and moves the file pointer to the next line. The `mysqli_fetch_row()` function is very similar in that it returns the fields in the current row of a resultset into an indexed array and moves the result pointer to the next row. You can then use the array to access the individual fields in the row. As an example, the following code prints the contents of the fields in the first row in the `inventory` table of the `guitars` database:

```
$SQLstring = "SELECT * FROM inventory";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
$Row = mysqli_fetch_row($QueryResult);
```

```
echo "<p><strong>Make</strong>: {$Row[0]}<br />";
echo "<strong>Model</strong>: {$Row[1]}<br />";
echo "<strong>Price</strong>: {$Row[2]}<br />";
echo "<strong>Quantity</strong>: {$Row[3]}</p>";
```

The `mysqli_fetch_row()` function returns the fields in the current row or a value of false when it reaches the last row in the resultset. This allows you to iterate through all the rows in a resultset. The following code shows a more complex example that uses a `do...while` statement to print all of the rows in the `inventory` table to an HTML table. Figure 9-6 shows how the table appears in a Web browser.

```
echo "<table width='100%' border='1'>";
echo "<tr><th>Make</th><th>Model</th>
     <th>Price</th><th>Quantity</th></tr>";
$Row = mysqli_fetch_row($QueryResult);
do {
     echo "<tr><td>{$Row[0]}</td>";
     echo "<td>{$Row[1]}</td>";
     echo "<td align='right'>{$Row[2]}</td>";
     echo "<td align='right'>{$Row[3]}</td></tr>";
     $Row = mysqli_fetch_row($QueryResult);
} while ($Row);
```


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapter.09/E:
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

Make	Model	Price	Quantity
Martin	D15 Spruce and Rosewood	1370.00	2
Washburn	D30s	799.99	5
Washburn	D100	329.90	10
Martin	D15 Limited Edition	1138.00	1
Fender	DG11	285.70	8
Martin	DX1 Dreadnought	699.00	9
Fender	DG7	368.20	14
Taylor	Baby Taylor Mahogany	348.00	7
Washburn	D10s	349.99	18
Yamaha	FG720S	279.99	3

Figure 9-6 Output of the `inventory` table in a Web browser

Next, you add query statements to the FlightlogEntries.php script that select all the records in the `flightsessions` table.

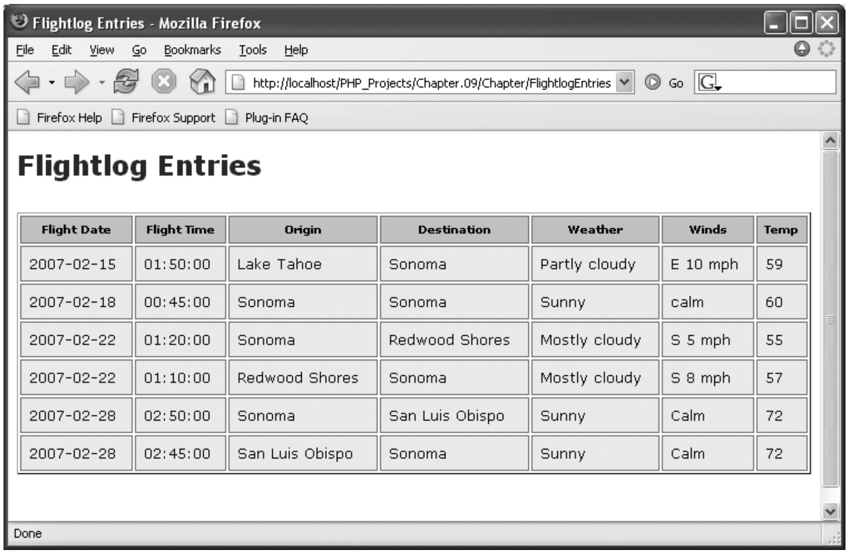
To add query statements to the FlightlogEntries.php script that select all the records in the `flightsessions` table:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Delete the following `echo()` statement that prints when the script successfully connects to the database server:

   ```
   echo "<p>Successfully connected to the database server.</p>";
   ```
3. Delete the following `echo()` statement that prints when the database opens successfully:

   ```
   echo "<p>Successfully opened the database.</p>";
   ```
4. Replace the statement that prints when the query executes successfully with the following statements, which use the `mysqli_fetch_row()` function to print the results in a table:

   ```
   echo "<table width='100%' border='1'>";
   echo "<tr><th>Flight Date</th><th>Flight Time</th>
   <th>Origin</th><th>Destination</th><th>Weather</th><th>Winds</th>
   <th>Temp</th></tr>";
   $Row = mysqli_fetch_row($QueryResult);
   do {
        echo "<tr><td>{$Row[0]}</td>";
        echo "<td>{$Row[1]}</td>";
        echo "<td>{$Row[2]}</td>";
        echo "<td>{$Row[3]}</td>";
        echo "<td>{$Row[4]}</td>";
        echo "<td>{$Row[5]}</td>";
        echo "<td>{$Row[6]}</td></tr>";
        $Row = mysqli_fetch_row($QueryResult);
   } while ($Row);
   ```
5. Save the **FlightlogEntries.php** file and open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. Your Web page should be similar to Figure 9-7, although you may have added or deleted additional entries.


Flightlog Entries - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

Flightlog Entries

Flight Date	Flight Time	Origin	Destination	Weather	Winds	Temp
2007-02-15	01:50:00	Lake Tahoe	Sonoma	Partly cloudy	E 10 mph	59
2007-02-18	00:45:00	Sonoma	Sonoma	Sunny	calm	60
2007-02-22	01:20:00	Sonoma	Redwood Shores	Mostly cloudy	S 5 mph	55
2007-02-22	01:10:00	Redwood Shores	Sonoma	Mostly cloudy	S 8 mph	57
2007-02-28	02:50:00	Sonoma	San Luis Obispo	Sunny	Calm	72
2007-02-28	02:45:00	San Luis Obispo	Sonoma	Sunny	Calm	72

Figure 9-7 Output of FlightlogEntries.php with the `mysqli_fetch_row()` function

6. Close your Web browser window.

Retrieving Records into an Associative Array

The `mysqli_fetch_assoc()` function returns the fields in the current row of a resultset into an associative array and moves the result pointer to the next row. The primary difference between the `mysqli_fetch_assoc()` function and the `mysqli_fetch_row()` function is that instead of returning the fields into an indexed array, the `mysqli_fetch_assoc()` function returns the fields into an associate array and uses each field name as the array key. For example, the following code uses the `mysqli_fetch_assoc()` function to print the contents of the fields in the first row in the `inventory` table of the `guitars` database. Notice that the `echo()` statements refer to keys instead of indexes in the `$Row[]` array.

```
$Row = mysqli_fetch_assoc($QueryResult);
echo "<p><strong>Make</strong>: {$Row['make']}<br />";
echo "<strong>Model</strong>: {$Row['model']}<br />";
echo "<strong>Price</strong>: {$Row['price']}<br />";
echo "<strong>Quantity</strong>: {$Row['quantity']}</p>";
```

The following code shows an associative array version of the `do...while` statement that prints all of the rows in the `inventory` table to an HTML table:

```
echo "<table width='100%' border='1'>";
echo "<tr><th>Make</th><th>Model</th>
<th>Price</th><th>Quantity</th></tr>";
do {
     $Row = mysqli_fetch_assoc($QueryResult);
     echo "<tr><td>{$Row['make']}</td>";
     echo "<td>{$Row['model']}</td>";
     echo "<td align='right'>{$Row['price']}</td>";
     echo "<td align='right'>{$Row['quantity']}</td></tr>";
} while ($Row);
```

Next, you add query statements to the FlightlogEntries.php script that select all the records in the `flightsessions` table.

To query statements to the FlightlogEntries.php script that select all the records in the `flightsessions` table:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Replace the two `mysqli_fetch_row()` functions with `mysqli_fetch_assoc()` functions.
3. Modify the `echo()` statements in the `do...while` statement so they reference the keys in the associative array instead of the index values. Your modified code should appear as follows:

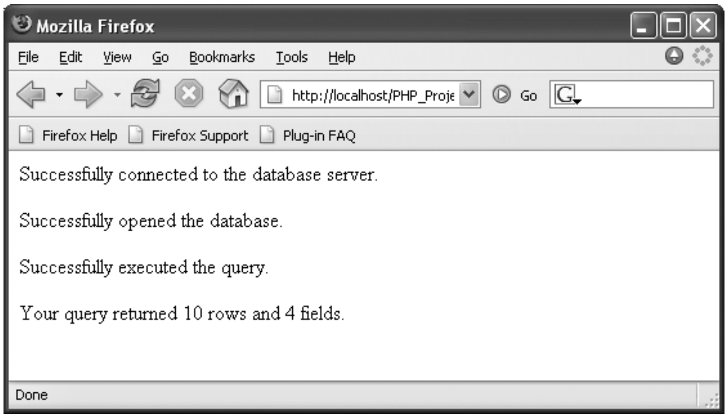
```
echo "<table width='100%' border='1'>";
echo "<tr><th>Flight Date</th><th>Flight Time</th>
<th>Origin</th><th>Destination</th><th>Weather</th><th>Winds</th>
<th>Temp</th></tr>";
$Row = mysqli_fetch_assoc($QueryResult);
do {
     echo "<tr><td>{$Row['flight_date']}</td>";
     echo "<td>{$Row['flight_time']}</td>";
     echo "<td>{$Row['origin']}</td>";
     echo "<td>{$Row['destination']}</td>";
     echo "<td>{$Row['weather']}</td>";
     echo "<td>{$Row['winds']}</td>";
     echo "<td>{$Row['temp']}</td></tr>";
     $Row = mysqli_fetch_assoc($QueryResult);
} while ($Row);
```

4. Save the **FlightlogEntries.php** file and open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. Your Web page should appear the same as it did before you modified the code to use `mysqli_fetch_assoc()` functions.
5. Close your Web browser window.

Accessing Query Result Information

PHP includes numerous functions for working with query results, including the `mysqli_num_rows()` function, which returns the number of rows in a query result, and the `mysqli_num_fields()` function, which returns the number of fields in a query result. Both functions accept a database connection variable as an argument. The following code demonstrates how to use both functions with the query results returned from the `guitars` database. If the number of rows and fields in the query result are not equal to zero, an `echo()` statement prints the number of rows and fields. However, if the number of rows and fields in the query result are equal to zero, an `echo()` statement prints "Your query returned no results." Figure 9-8 shows the output if the `guitars` database contains 10 rows and 4 fields.

```
$SQLstring = "SELECT * FROM inventory";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully executed the query.</p>";
$NumRows = mysqli_num_rows($QueryResult);
$NumFields = mysqli_num_fields($QueryResult);
if ($NumRows != 0 && $NumFields != 0)
     echo "<p>Your query returned " .
mysqli_num_rows($QueryResult) . " rows and "
     . mysqli_num_fields($QueryResult) . " fields.</p>";
else
     echo "<p>Your query returned no results.</p>";
mysqli_close($DBConnect);
```


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Proje
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

```
Successfully connected to the database server.

Successfully opened the database.

Successfully executed the query.

Your query returned 10 rows and 4 fields.
```

Figure 9-8 Output of the number of rows and fields returned from a query

Next, you add statements to the FlightlogEntries.php script that print the number of returned rows and fields.

To add statements to the FlightlogEntries.php script that print the number of returned rows and fields:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Add the following statements above the above the `mysqli_close($DBConnect);` statement:

   ```
   $NumRows = mysqli_num_rows($QueryResult);
   $NumFields = mysqli_num_fields($QueryResult);
   echo "<p>Your query returned the following "
        . mysqli_num_rows($QueryResult)
        . " rows and ". mysqli_num_fields($QueryResult)
        . " fields:</p>";
   ```

3. Save the **FlightlogEntries.php** file and open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. Your Web page should appear the same as it did before you added modified the code to use `mysqli_fetch_assoc()` functions.
4. Close your Web browser window.

Closing Query Results

When you are finished working with query results retrieved with the `mysqli_query()` function, you should use the `mysqli_free_result()` function to close the resultset. This ensures that the resultset doesn't keep taking up space in your computer's memory. (As you'll recall, you need to close a database connection for the same reason.) To close the resultset, pass to the `mysqli_free_result()` function the variable containing the result pointer from the `mysqli_query()` function. The following code uses the `mysqli_free_result()` function to close the `$QueryResult` variable:

```
$SQLstring = "SELECT * FROM inventory";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully executed the query.</p>";
...
mysqli_free_result($QueryResult);
mysqli_close($DBConnect);
```

CAUTION

You can only use the `mysqli_free_result()` function with SQL statements that return results, such as `SELECT` queries. If you attempt to use the `mysqli_free_result()` function with SQL statements that do not return results, such as the `CREATE DATABASE` and `CREATE TABLE` statements, you receive an error.

Next, you add a `mysqli_free_result()` function to the FlightlogEntries.php script.

To add a `mysqli_free_result()` function to the FlightlogEntries.php script:

1. Return to the **FlightlogEntries.php** document in your text editor.
2. Add the following statement above the `mysqli_close()` statement:

   ```
   mysqli_free_result($QueryResult);
   ```
3. Save the **FlightlogEntries.php** file and close it in your text editor. Then open the script in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/FlightlogEntries.php**. Your Web page should appear the same as it did before you added the `mysqli_free_result()` function.
4. Close your Web browser window.

Working with Databases and Tables

In this section, you learn how to use PHP to work with MySQL databases and tables. More specifically, you learn how to create and delete databases and tables. Again, keep in mind that the SQL statements in this section are identical to the SQL statements you saw in Chapter 8. The only difference is that they are executed with PHP instead of with the MySQL Monitor.

NOTE

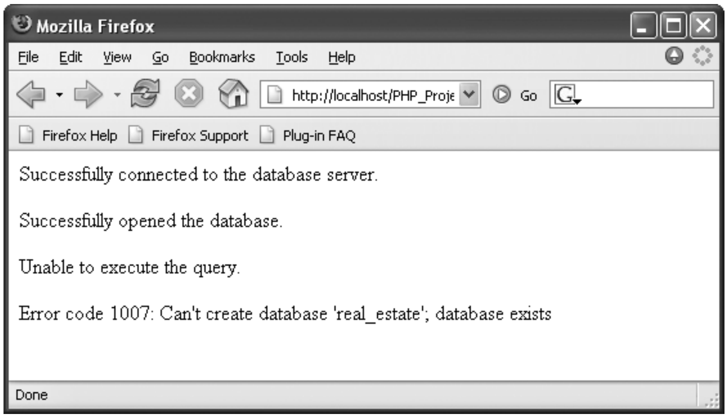
For information that you want to store permanently, you should use the MySQL Monitor instead of PHP to create and delete databases and tables. Creating and deleting databases and tables with PHP is most useful when you only need to temporarily store information for the current Web browser session.

Creating and Deleting Databases

You use the `CREATE DATABASE` statement with the `mysqli_query()` function to create a new database. The following statements create a database named `real_estate`:

```
$SQLstring = "CREATE DATABASE real_estate";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully executed the query.</p>";
mysqli_close($DBConnect);
```

If the `mysqli_query()` function successfully creates the database, you see the "Successfully executed the query" message shown in the preceding example. If the database already exists, you see the error code and message shown in Figure 9-9.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Proje
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

```
Successfully connected to the database server.

Successfully opened the database.

Unable to execute the query.

Error code 1007: Can't create database 'real_estate'; database exists
```

Figure 9-9 Error code and message that prints when you attempt to create a database that already exists

To avoid the error message shown in Figure 9-9, you should use the `mysqli_db_select()` function to check whether a database exists before you create or delete it. The following code attempts to select the real_estate database with the `mysqli_db_select()` function. Notice that the `mysqli_db_select()` function is preceded by the error control operator to suppress errors. If the `mysqli_db_select()` function successfully selects the `real_estate` database, the message "The real_estate database already exists!" prints to the Web browser. Otherwise, the statements in the `else` clause create the database.

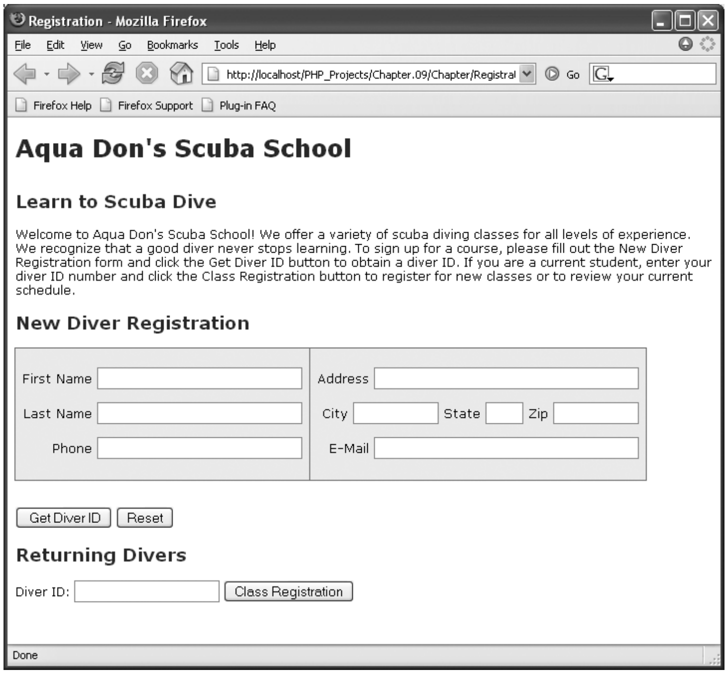
```
$DBName = "real_estate";
if (@mysqli_select_db($DBConnect, $DBName))
     echo "<p>The $DBName database already exists!</p>";
else {
     $SQLstring = "CREATE DATABASE $DBName";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
           Or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>Successfully created the database.</p>";
mysqli_select_db($DBConnect, $DBName)
}
mysqli_close($DBConnect);
```

As with the MySQL Monitor, creating a new database does not select it. To use a new database, you must select it by executing the `mysqli_select_db()` function. The `real_estate` database is selected at the end of the `else` clause in the preceding code.

Deleting a database is almost identical to creating one, except that you use the `DROP DATABASE` statement instead of the `CREATE DATABASE` statement with the `mysqli_query()` function. The following code demonstrates how to delete the `real_estate` database. Notice that the code uses the same error-handling functionality as the code that created the database.

```
$DBName = "real_estate";
...
if (@!mysqli_select_db($DBConnect, $DBName))
     echo "<p>The $DBName database does not exist!</p>";
else {
     $SQLstring = "DROP DATABASE $DBName";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
           Or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>Successfully deleted the database.</p>";
}
mysqli_close($DBConnect);
```

In the rest of this chapter, you work on a Web site for registering students in scuba diving classes for a company named Aqua Don's Scuba School. Student information and class registrations will be stored in a MySQL database named `scuba_school` consisting of two tables: `divers` and `registration`. The `divers` table contains each diver's ID, along with other personal information. The `registration` table contains a record for each class in which a diver enrolls. The `divers` table is the primary table, and the `diverID` field acts as the primary key. The `diverID` field also acts as the foreign key in the `registration` table. Because each student can enroll in more than one class, the relationship between the `students` table and the `registration` table is one-to-many; the `students` table is the one side of the relationship, and the `registration` table is the many side of the relationship. Your Chapter directory for Chapter 9 contains a document named Registration.html that you will use to call some PHP scripts that access the MySQL database. Figure 9-10 shows the Registration.html page in a Web browser.


Registration - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapter.09/Chapter/Registral
Go
Firefox Help Firefox Support Plug-in FAQ

Aqua Don's Scuba School

Learn to Scuba Dive

Welcome to Aqua Don's Scuba School! We offer a variety of scuba diving classes for all levels of experience. We recognize that a good diver never stops learning. To sign up for a course, please fill out the New Diver Registration form and click the Get Diver ID button to obtain a diver ID. If you are a current student, enter your diver ID number and click the Class Registration button to register for new classes or to review your current schedule.

New Diver Registration

First Name		Address	
Last Name		City / State / Zip	
Phone		E-Mail	

Get Diver ID   Reset

Returning Divers

Diver ID: [ ] Class Registration

Figure 9-10 Registration.html page in a Web browser

First, you create a script named GetDiverID.php that registers divers with Aqua Don's Scuba School. You add code to the GetDiverID.php script that creates the `scuba_school` database the first time the script is called.

To create the GetDiverID.php script:

1. Create a new document in your text editor.
2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and `<body>` element. Use the strict DTD and "Register Diver" as the content of the `<title>` element.
3. Add the following `<link>` element above the closing `</head>` tag to link to the php_styles.css style sheet in your Chapter directory:

   ```
   <link rel="stylesheet" href="php_styles.css" type="text/css" />
   ```

4. Add the following heading element to the document body:

   ```
   <h1>Aqua Don's Scuba School Registration</h1>
   ```

5. Add the following script section to the end of the document body:

   ```
   <?php
   ?>
   ```

6. Add the following statements to the script section to ensure that users enter all the fields in New Diver Registration form:

```
if (empty($_GET['first_name']) || empty($_GET['last_name']) ||
empty($_GET['phone']) || empty($_GET['address']) ||
empty($_GET['city']) || empty($_GET['state']) ||
empty($_GET['zip']) || empty($_GET['email']))
     exit("<p>You must enter values in all fields of the New
Diver Registration form! Click your browser's Back button to
return to the previous page.</p>");
```

7. Add the following statements to the end of the script section to connect to the database server. Replace *user* and *password* with the MySQL username and password you created in Chapter 8.

```
$DBConnect = @mysqli_connect("localhost", "user", "password")
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
```

8. Add the following statements, which create and select the `scuba_school` database. The contents of the conditional expression in the `if` statement only execute if the `mysqli_select_db()` function returns a value of false, which means the database does not exist. Because the contents of the `if` statement only execute the first time you open the script, the "Successfully created the database" message only appears once.

```
$DBName = "scuba_school";
if (!@mysqli_select_db($DBConnect, $DBName)) {
     $SQLstring = "CREATE DATABASE $DBName";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
           Or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>Successfully created the database.</p>";
     mysqli_select_db($DBConnect, $DBName);
}
```

9. Add the following statement to the end of the script section to close the database connection:

```
mysqli_close($DBConnect);
```

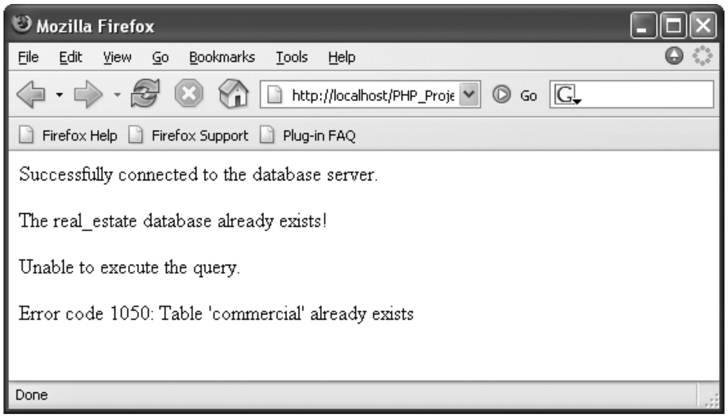
10. Save the document as **GetDiverID.php** in the Chapter directory for Chapter 9.

Creating and Deleting Tables

To create a table, you use the `CREATE TABLE` statement with the `mysqli_query()` function. Be sure you have executed the `mysqli_select_db()` function before executing the `CREATE TABLE` statement or you might create your new table in the wrong database. The following code creates a table named `commercial` in the `real_estate` database.

```
$DBName = "real_estate";
...
$SQLstring = "CREATE TABLE commercial (city VARCHAR(25), state
VARCHAR(25), sale_or_lease VARCHAR(25), type_of_use VARCHAR(40),
price INT, size INT)";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully created the table.</p>";
mysqli_close($DBConnect);
```

With the preceding code, if the table already exists in the selected database, you will see the error code and message shown in Figure 9-11.


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Proje
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

```
Successfully connected to the database server.

The real_estate database already exists!

Unable to execute the query.

Error code 1050: Table 'commercial' already exists
```

Figure 9-11 Error code and message that prints when you attempt to create a table that already exists

To prevent your code from attempting to create a table that already exists, use a **`mysqli_query()`** function that attempts to select records from the table. If the function executes successfully and returns a value of true, the table already exists. The following code demonstrates how to check whether a table exists before attempting to create it:

```
$DBName = "real_estate";
...
$TableName = "commercial";
$SQLstring = "SELECT * FROM $TableName";
$QueryResult = @mysqli_query($DBConnect, $SQLstring);
```

```
if ($QueryResult)
     echo "<p>The $TableName table already exists!</p>";
else {
     $SQLstring = "CREATE TABLE commercial (city VARCHAR(25),
     state VARCHAR(25), sale_or_lease VARCHAR(25),
     type_of_use VARCHAR(40), price INT, size INT)";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
           Or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>Successfully created the table.</p>";
}
 mysqli_close($DBConnect);
```

Next, you add code to the GetDiverID.php script that creates the `divers` table the first time the script is called. The `divers` table will use the `diverID` field as the primary key. To identify a field as a primary key in MySQL, you include the `PRIMARY KEY` keywords when you first define a field with the `CREATE TABLE` statement. The `AUTO_INCREMENT` keyword is often used with a primary key to generate a unique ID for each new row in a table. The first row in a field that is created with the `AUTO_INCREMENT` keyword is assigned a value of 1. The value for each subsequently added row is incremented by 1 from the preceding row. Another keyword that is often used with primary keys is the `NOT NULL` keyword, which requires a field to include a value. As an example, the following SQL statement defines a primary key named `id` for the `inventory` table using the `SMALLINT` data type. The `id` field definition also includes the `NOT NULL` and `AUTO_INCREMENT` keywords.

```
CREATE TABLE inventory (id SMALLINT NOT NULL AUTO_INCREMENT
PRIMARY KEY, make VARCHAR(25), model VARCHAR(50), price FLOAT,
quantity INT);
```

When you add records to a table that includes an `AUTO_INCREMENT` field, you specify `NULL` as the field value. The following SQL statement inserts a new record into the `inventory` table of the `guitars` database. If this is the first record added to the table, its primary key will be a value of 1.

```
INSERT INTO inventory VALUES(NULL, 'Ovation',
'1777 LX Legend', 1049.00, NULL);
```

Next, you add code to the GetDiverID.php script that creates the `divers` table the first time the script is called. The `divers` table includes an autoincrementing primary key.

To add code to the GetDiverID.php script that creates the `divers` table the first time the script is called:

1. Return to the **GetDiverID.php** document in your text editor.

2. Add the following variable declarations and `mysqli_query()` statement to the end of the script section. The `mysqli_query()` statement selects all existing records from the `divers` table.

   ```
   $TableName = "divers";
   $SQLstring = "SELECT * FROM $TableName";
   $QueryResult = @mysqli_query($DBConnect, $SQLstring);
   ```

3. Add the following `if` statement to the end of the script section. The statements in the `if` statement only execute if the `$QueryResult` variable contains a value of false, which means that it does not yet exist. Notice that the `CREATE TABLE` statement creates the `diverID` field as an autoincrementing primary key.

```
if (!$QueryResult) {
     $SQLstring = "CREATE TABLE divers (diverID SMALLINT NOT
          NULL AUTO_INCREMENT PRIMARY KEY, first VARCHAR(40),
          last VARCHAR(40), phone VARCHAR(40),
          address VARCHAR(40), city VARCHAR(40),
          state VARCHAR(2), zip VARCHAR(10))";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
          Or die("<p>Unable to create the divers table.</p>"
          . "<p>Error code " . mysqli_errno($DBConnect)
          . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>Successfully created the divers table.</p>";
}
```

4. Save the **GetDiverID.php** document.

To delete a table, you use the `DROP TABLE` statement with the `mysqli_query()` function. The following code demonstrates how to delete the `commercial` table using similar error-handling functionality as the code that created the table:

```
$DBName = "real_estate";
...
$TableName = "commercial";
$SQLstring = "SELECT * FROM $TableName";
$QueryResult = @mysqli_query($DBConnect, $SQLstring);
if (!$QueryResult)
     echo "<p>The $TableName table does not exist!</p>";
else {
     $SQLstring = "DROP TABLE commercial";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
           Or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>Successfully deleted the table.</p>";
}
mysqli_close($DBConnect);
```

MANIPULATING RECORDS

In this section, you learn how to use PHP to add, update, and delete database records.

Adding, Deleting, and Updating Records

To add records to a table, you use the INSERT and VALUES keywords with the mysqli_query() function. Remember that the values you enter in the VALUES list must be in the same order in which you defined the table fields. For example, the following statements add a new row to the inventory table in the guitars database:

```
$SQLstring = "INSERT INTO inventory VALUES('Ovation',
      '1777 LX Legend', 1049.00, 2)";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
      Or die("<p>Unable to execute the query.</p>"
      . "<p>Error code " . mysqli_errno($DBConnect)
      . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully added the record.</p>";
```

Also remember that you must specify NULL in any fields for which you do not have a value. For example, if you do not know the quantity of guitars in stock for the Ovation guitar, you can enter NULL as the last item in the VALUES list, as follows:

```
$SQLstring = "INSERT INTO inventory VALUES('Ovation',
      '1777 LX Legend', 1049.00, NULL)";
```

To add multiple records to a database, you use the LOAD DATA statement and the mysqli_query() function with a local text file containing the records you want to add. The following statement loads a file named inventory.txt into the inventory table in the guitars database:

```
$SQLstring = "LOAD DATA LOCAL INFILE 'inventory.txt'
     INTO TABLE inventory";
```

To update records in a table, you use the UPDATE, SET, and WHERE keywords with the mysqli_query() function. The UPDATE keyword specifies the name of the table to update and the SET keyword specifies the value to assign to the fields in the records that match the condition in the WHERE keyword. For example, the following statements modify the price of the Fender DG7 guitar to $368.20:

```
$SQLstring = "UPDATE inventory SET price=368.20
     WHERE make='Fender' AND model='DG7'";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully modified the records.</p>";
```

To delete records in a table, you use the DELETE and WHERE keywords with the mysqli_query() function. Remember that the WHERE keyword determines which records to delete in the table. For example, the following statement deletes the "Taylor 210 Dreadnought" record from the inventory table in the guitars database:

```
$SQLstring = "DELETE FROM inventory WHERE make='Taylor'
     AND model='210 Dreadnought'";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully deleted the records.</p>";
```

To delete all the records in a table, omit the WHERE keyword. For example, the following statement deletes all the records in the inventory table:

```
$SQLstring = "DELETE FROM inventory";
```

Next, you add code to the GetDiverID.php script that adds a new diver record to the divers table in the scuba_school database. You also use the mysqli_insert_id() function, which returns the ID created with AUTO_INCREMENT in the last INSERT operation. You pass to the mysqli_insert_id() function the variable to which you assigned the database connection with the mysqli_connect() function. The mysqli_insert_id() function is useful when you need to find the primary key created for new records you add to a database table.

To add code to the GetDiverID.php script that adds a new diver record to the divers table in the scuba_school database:

1. Return to the **GetDiverID.php** document in your text editor.
2. Add the following statements above the mysqli_close() statement to copy the values that were passed from the form in the Registration.html to PHP variables:

```
$First = addslashes($_GET['first_name']);
$Last = addslashes($_GET['last_name']);
$Phone = addslashes($_GET['phone']);
$Address = addslashes($_GET['address']);
$City = addslashes($_GET['city']);
$State = addslashes($_GET['state']);
$Zip = addslashes($_GET['zip']);
$Email = addslashes($_GET['email']);
```

3. Add the following statements above the `mysqli_close()` statement to build a query string that will insert the values into the `diver` table:

```
$SQLstring = "INSERT INTO divers VALUES(NULL, '$First', '$Last',
'$Phone', '$Address', '$City', '$State', '$Zip')";
```

4. Add the following statements above the `mysqli_close()` statement to execute the query:

```
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
```

5. Add the following `mysqli_insert_id()` statement above the `mysqli_close()` statement to assign the new primary key to the `$DiverID` variable:

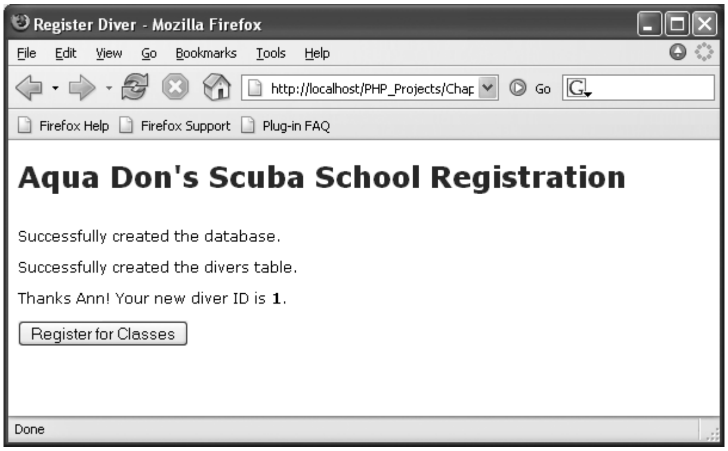
```
$DiverID = mysqli_insert_id($DBConnect);
```

6. Finally, add the following text and elements to the end of the document body. The form allows users to register for classes by clicking the Register for Classes button, which opens a script named CourseListings.php. Notice that the form includes a hidden variable that is assigned the value of the `$DiverID` variable. This ensures that the diver ID is passed to the CourseListings.php script when the user clicks the Register for Classes button.

```
<p>Thanks <?= $First ?>! Your new diver ID is <strong><?=
$DiverID ?></strong>.</p>
<form action="CourseListings.php" method="get">
<p><input type="submit" value="Register for Classes" />
<input type="hidden" name="diverID" value="<?= $DiverID ?>"
/></p>
</form>
```

7. Save the **GetDiverID.php** document and close it in your text editor.

8. Open the **Registration.html** file in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/Registration.html**. Enter values into the New Diver Registration form and click the **Get Diver ID** button. You should be assigned a new diver ID of 1. You should see the Web page shown in Figure 9-12.


Register Diver - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chap
Go
Firefox Help Firefox Support Plug-in FAQ
Register for Classes
Done

Aqua Don's Scuba School Registration

Successfully created the database.

Successfully created the divers table.

Thanks Ann! Your new diver ID is **1**.

Figure 9-12 Register Diver Web page

9. Click your browser's Back button, enter some new values in the New Diver Registration form, and then click the **Get Diver ID** button. The new diver ID should be 2. Notice that the messages about successfully creating the database and `divers` table do not appear this time.
10. Close your Web browser window.

Next, you create the CourseListings.php script, which divers can use to register for classes.

To create the CourseListings.php script:

1. Create a new document in your text editor.
2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and `<body>` element. Use the strict DTD and "Course Listings" as the content of the `<title>` element.
3. Add the following `<link>` element above the closing `</head>` tag to link to the php_styles.css style sheet in your Chapter directory:

```
<link rel="stylesheet" href="php_styles.css" type="text/css" />
```

4. Add the following heading element to the document body:

```
<h1>Aqua Don's Scuba School</h1>
<h2>Class Registration Form</h2>
```

5. Add the following script section to the end of the document body:

```
<?php
?>
```

6. Add the following statements to the script section to connect to the database server and open the `scuba_school` database. Replace *`user`* and *`password`* with the MySQL username and password you created in Chapter 8.

```
$DBConnect = @mysqli_connect("localhost", "user", "password")
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
$DBName = "scuba_school";
@mysqli_select_db($DBConnect, $DBName)
     Or die("<p>Unable to select the database.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
```

7. Add the following statements to the end of the script section to ensure that users open the page with a valid diver ID:

```
$DiverID = $_GET['diverID'];
if (empty($DiverID))
     exit("<p>You must enter a diver ID! Click your browser's
Back button to return to the previous page.</p>");
$TableName = "divers";
$SQLstring = "SELECT * FROM $TableName WHERE diverID='$DiverID'";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
if (mysqli_num_rows($QueryResult) == 0)
     die("<p>You must enter a valid diver ID! Click your
browser's Back button to return to the Registration form.</p.");
```

8. Add the following statement to the end of the script section to close the database connection:

```
mysqli_close($DBConnect);
```

9. Add the following form to the end of document body. This form allows divers to review their current schedule with the ReviewSchedule.php script.

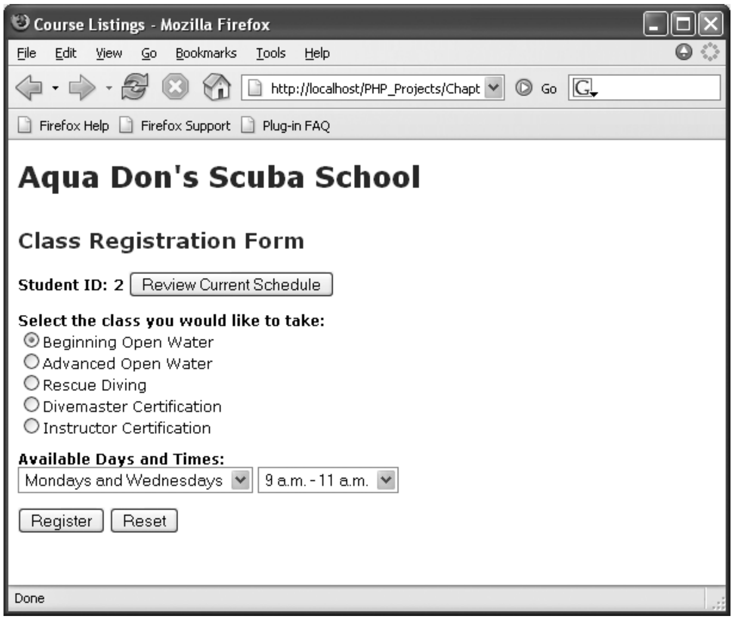
```
<form method="get" action="ReviewSchedule.php">
<p><strong>Student ID: <?= $DiverID ?></strong>
<input type="submit" value=" Review Current Schedule " /><input
type="hidden" name="diverID" value="<?= $DiverID ?>" /></p>
</form>
```

10. Add the following form to the end of document body. This is the form divers use to register for classes with the RegisterDiver.php script.

```
<form method="get" action="RegisterDiver.php">
<p><strong>Select the class you would like to take:</strong><br />
<input type="radio" name="class" value="Beginning Open Water"
checked="checked" />Beginning Open Water<br />
<input type="radio" name="class" value="Advanced Open Water" />
Advanced Open Water<br />
<input type="radio" name="class" value="Rescue Diving" />
Rescue Diving<br />
<input type="radio" name="class"
value="Divemaster Certification" />Divemaster Certification<br />
<input type="radio" name="class"
value="Instructor Certification" />Instructor Certification</p>
<p><strong>Available Days and Times:</strong><br />
<select name="days">
<option selected="selected" value="Mondays and Wednesdays">
Mondays and Wednesdays</option>
<option value="Tuesdays and Thursdays">
Tuesdays and Thursdays</option>
<option value="Wednesdays and Fridays">
Wednesdays and Fridays</option>
</select>
<select name="time">
<option selected="selected" value="9 a.m. - 11 a.m.">9 a.m. - 11
a.m.</option>
<option value="1 p.m. - 3 p.m.">1 p.m. - 3 p.m.</option>
<option value="6 p.m. - 8 p.m.">6 p.m. - 8 p.m.</option>
</select><input type="hidden" name="diverID"
value="<?= $DiverID ?>" /></p>
<p><input type="submit" value=" Register " />
<input type="reset" /></p>
</form>
```

11. Save the document as **CourseListings.php** in the Chapter directory for Chapter 9, and then close it in your text editor.

12. Open the **Registration.html** file in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Chapter/Registration.html**. Enter an existing diver ID into the Returning Divers form and click the **Class Registration** button. You should see the Web page shown in Figure 9-13.


Course Listings - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapt
Go
Firefox Help Firefox Support Plug-in FAQ
Review Current Schedule
Register
Reset
Done

Aqua Don's Scuba School

Class Registration Form

Student ID: 2

Select the class you would like to take:

[✓ Beginning Open Water]
Advanced Open Water
Rescue Diving
Divemaster Certification
Instructor Certification

Available Days and Times:

Mondays and Wednesdays
9 a.m. - 11 a.m.

Figure 9-13 Course Listings Web page

13. Leave the Course Listings page open in your Web browser.

Next, you create the RegisterDiver.php script, which adds diver registration information to the `registration` table.

To create the RegisterDiver.php script:

1. Create a new document in your text editor.
2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and `<body>` element. Use the strict DTD and "Register Diver" as the content of the `<title>` element.
3. Add the following `<link>` element above the closing `</head>` tag to link to the php_styles.css style sheet in your Chapter directory:

```
<link rel="stylesheet" href="php_styles.css" type="text/css" />
```

4. Add the following heading element to the document body:

```
<h1>Aqua Don's Scuba School</h1>
<h2>Registration Confirmation</h2>
```

5. Add the following script section to the end of the document body:

```
<?php
?>
```

6. Add the following statements to the script section to ensure that users open the page with a valid diver ID:

```
$DiverID = $_GET['diverID'];
if (empty($DiverID))
     exit("<p>You must enter a diver ID! Click your browser's
Back button to return to the previous page.</p>");
```

7. Add the following statements to the end of the script section to connect to the database server and open the `scuba_school` database:

```
$DBConnect = @mysqli_connect("localhost", "dongosselin",
"rosebud")
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
$DBName = "scuba_school";
@mysqli_select_db($DBConnect, $DBName)
     Or die("<p>Unable to select the database.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
```

8. Add the following statements to the end of the script section to create the registration table if it does not exist:

```
$TableName = "registration";
$SQLstring = "SELECT * FROM $TableName";
$QueryResult = @mysqli_query($DBConnect, $SQLstring);
if (!$QueryResult) {
     $SQLstring = "CREATE TABLE registration (diverID SMALLINT,
class VARCHAR(40), days VARCHAR(40), time VARCHAR(40))";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
           Or die("<p>Unable to create the registration
table.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>Successfully created the registration table.</p>";
}
```

9. Add the following statements to the end of the script section to register the diver in the selected class:

```
$Class = $_GET['class'];
$Days = $_GET['days'];
$Time = $_GET['time'];
$SQLstring = "INSERT INTO $TableName VALUES('$DiverID', '$Class',
     '$Days', '$Time')";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
```

10. Add the following statement to the end of the script section to close the database connection:

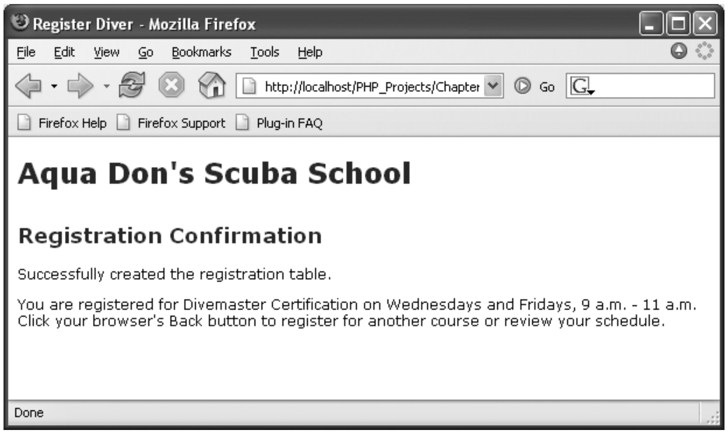
```
mysqli_close($DBConnect);
```

11. Finally, add the following text and elements to the end of the document body:

```
<p>You are registered for <?= "$Class on $Days, $Time" ?>. Click
your browser's Back button to register for another course or
review your schedule.</p>
```

12. Save the document as **RegisterDiver.php** in the Chapter directory for Chapter 9, and then close it in your text editor.

13. Return to the **Course Listings** page in your Web browser. Select a class, as well as the days and times you want to take it, and then click the **Register** button. You should see a message indicating that the `registration` table was created successfully, along with a message confirming your registration in the class, as shown in Figure 9-14.


Register Diver - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapter
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

Aqua Don's Scuba School

Registration Confirmation

Successfully created the registration table.

You are registered for Divemaster Certification on Wednesdays and Fridays, 9 a.m. - 11 a.m. Click your browser's Back button to register for another course or review your schedule.

Figure 9-14 Registration Confirmation Web page

14. Click your browser's **Back** button to return to the Course Listings page.

The last script you create is the ReviewSchedule.php script, which allows divers to review the classes in which they are registered.

To create the ReviewSchedule.php script:

1. Create a new document in your text editor.
2. Type the `<!DOCTYPE>` declaration, `<html>` element, header information, and `<body>` element. Use the strict DTD and "Review Schedule" as the content of the `<title>` element.

3. Add the following `<link>` element above the closing `</head>` tag to link to the php_styles.css style sheet in your Chapter directory:

```
<link rel="stylesheet" href="php_styles.css" type="text/css" />
```

4. Add the following heading element to the document body:

```
<h1>Aqua Don's Scuba School</h1>
<h2>This is your current schedule:</h2>
```

5. Add the following script section to the end of the document body:

```
<?php
?>
```

6. Add the following statements to the script section to ensure that users open the page with a valid diver ID:

```
$DiverID = $_GET['diverID'];
if (empty($DiverID))
     exit("<p>You must enter a diver ID! Click your browser's
Back button to return to the previous page.</p>");
```

7. Add the following statements to the end of the script section to connect to the database server and open the `scuba_school` database. Replace *`user`* and *`password`* with the MySQL username and password you created in Chapter 8.

```
$DBConnect = @mysqli_connect("localhost", "user", "password")
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
$DBName = "scuba_school";
@mysqli_select_db($DBConnect, $DBName)
     Or die("<p>Unable to select the database.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
```

8. Add the following statements to the end of the script section to query the database for all records that match the diver ID:

```
$TableName = "registration";
$SQLstring = "SELECT * FROM $TableName WHERE diverID='$DiverID'";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
```

9. Next, add the following statements to the end of the script section, which print a message if the diver has not yet registered for any classes:

```
if (mysqli_num_rows($QueryResult) == 0)
     die("<p>You have not registered for any classes! Click your
     browser's Back button to return to the previous page.</p>");
```

10. Add the following statements to the end of the script section to print the results in an HTML table:

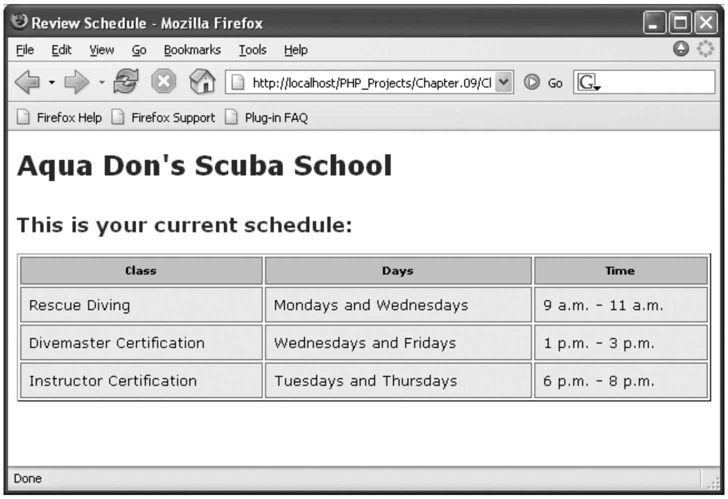
```
echo "<table width='100%' border='1'>";
echo "<tr><th>Class</th><th>Days</th>
<th>Time</th></tr>";
$Row = mysqli_fetch_assoc($QueryResult);
do {
     echo "<tr><td>{$Row['class']}</td>";
     echo "<td>{$Row['days']}</td>";
     echo "<td>{$Row['time']}</td></tr>";
     $Row = mysqli_fetch_assoc($QueryResult);
} while ($Row);
```

11. Finally, add the following statements to the end of the script section to close the database connection and the query results:

```
mysqli_free_result($QueryResult);
mysqli_close($DBConnect);
```

12. Save the document as **ReviewSchedule.php** in the Chapter directory for Chapter 9, and then close it in your text editor.

13. Return to the **Course Listings** page in your Web browser and register for several other classes. After you have registered for a few classes, click the Review Current Schedule button to display your schedule. Figure 9-15 shows the Review Schedule Web page for a diver who is signed up for three classes.


Review Schedule - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/PHP_Projects/Chapter.09/Cl
Go
Firefox Help
Firefox Support
Plug-in FAQ
Done

Aqua Don's Scuba School

This is your current schedule:

Class	Days	Time
Rescue Diving	Mondays and Wednesdays	9 a.m. - 11 a.m.
Divemaster Certification	Wednesdays and Fridays	1 p.m. - 3 p.m.
Instructor Certification	Tuesdays and Thursdays	6 p.m. - 8 p.m.

Figure 9-15 Review Schedule Web page

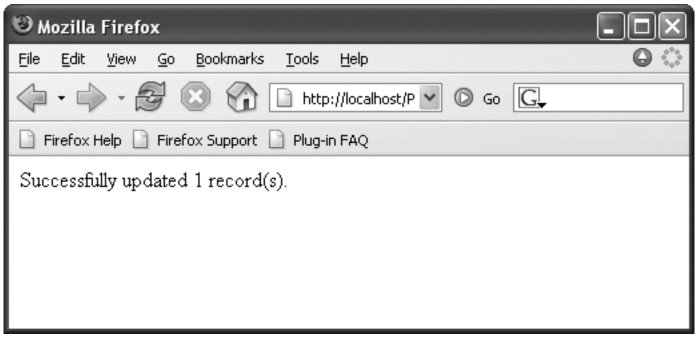
Returning Information on Affected Records

As you have learned, the `mysqli_num_rows()` function returns the number of rows in a query result and the `mysqli_num_fields()` function returns the number of fields in a query result. In addition, PHP includes two functions, `mysqli_affected_rows()` and `mysqli_info()`, which you can use to return information on the records that were affected by a query. First, you learn how to use the `mysqli_affected_rows()` function.

Using the `mysqli_affected_rows()` Function

With queries that return results, such as `SELECT` queries, you can use the `mysqli_num_rows()` function to find the number of records returned from the query. However, with queries that modify tables but do not return results, such as `INSERT`, `UPDATE`, and `DELETE` queries, you can use the `mysqli_affected_rows()` function to determine the number of affected rows.You pass to the `mysqli_affected_rows()` function the variable containing the database connection returned from the `mysqli_connect()` function—not the variable containing the result pointer from the `mysqli_query()` function. For example, the following statements print the number of rows affected by an `UPDATE` query. Figure 9-16 shows the output in a Web browser.

```
$SQLstring = "UPDATE inventory SET price=368.20
     WHERE make='Fender' AND model='DG7'";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully updated "
     . mysqli_affected_rows($DBConnect) . " record(s).</p>";
```



Successfully updated 1 record(s).

Figure 9-16 Output of `mysqli_affected_rows()` function for an `UPDATE` query

The following code contains another example of the `mysqli_affected_rows()` function, this time with a `DELETE` query:

```
$SQLstring = "DELETE FROM inventory WHERE make='Washburn'";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully deleted "
     . mysqli_affected_rows($DBConnect) . " record(s).</p>";
```

Using the `mysqli_info()` Function

For queries that add or update records, or that alter a table's structure, you can use the `mysqli_info()` function to return information about the query. The `mysqli_info()` function returns the number of operations for various types of actions, depending on the type of query. For example, with `INSERT` queries, the `mysqli_info()` function returns the number of records added and duplicated, along with the number of warnings. However, for `LOAD DATA` queries, the `mysqli_info()` function returns the number of records added, deleted, and skipped, along with the number of warnings. As with the `mysqli_affected_rows()` function, you pass to the `mysqli_info()` function the variable containing the database connection from the `mysqli_connect()` function. The `mysqli_info()` function returns information about the last query that was executed on the database connection. However, the `mysqli_info()` function returns information about queries that match one of the following formats:

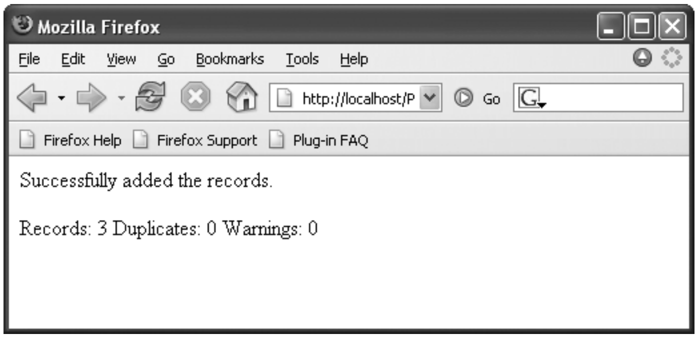
- `INSERT INTO...SELECT...`
- `INSERT INTO...VALUES (...),(...),(...)`
- `LOAD DATA INFILE ...`
- `ALTER TABLE ...`
- `UPDATE ...`

For any queries that do not match one of the preceding formats, the `mysqli_info()` function returns an empty string. Notice that the format for adding records with the `INSERT` and `VALUES` keywords includes multiple value sets. The `mysqli_info()` function only returns query information when you add multiple records with the `INSERT` keyword. For example, the `mysqli_info()` function in the following example returns an empty string because the `INSERT` query only adds a single record:

```
$SQLstring = "INSERT INTO inventory VALUES('Ovation',
      '1777 LX Legend', 1049.00, 2)";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
      Or die("<p>Unable to execute the query.</p>"
      . "<p>Error code " . mysqli_errno($DBConnect)
      . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully added the record.</p>";
echo "<p>" . mysqli_info($DBConnect) . "</p>";
```

In comparison, the following statements print the query information shown in Figure 9-17 because the `INSERT` query adds multiple records:

```
$SQLstring = "INSERT INTO inventory
     VALUES('Ovation', '1777 LX Legend', 1049.00, 2),
     ('Ovation', '1861 Standard Balladeer', 699.00, 1),
     ('Ovation', 'Tangent Series T357', 569.00, 3)";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully added the records.</p>";
echo "<p>" . mysqli_info($DBConnect) . "</p>";
```



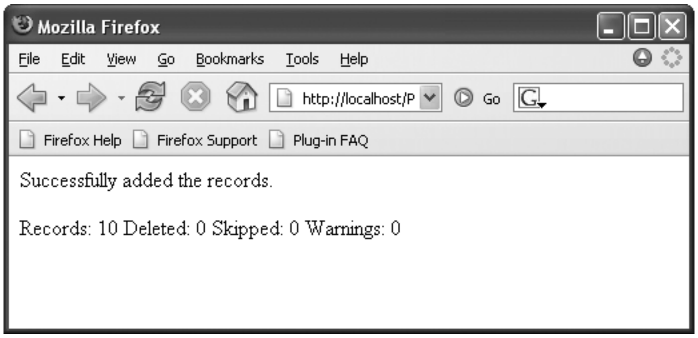
```
Successfully added the records.

Records: 3 Duplicates: 0 Warnings: 0
```

Figure 9-17 Output of `mysqli_info()` function for an `INSERT` query that adds multiple records

The `mysqli_info()` function also returns information for `LOAD DATA` queries. The following statements print the output shown in Figure 9-18:

```
$SQLstring = "LOAD DATA LOCAL INFILE 'c:/temp/inventory.txt'
     INTO TABLE inventory;";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<p>Successfully added the records.</p>";
echo "<p>" . mysqli_info($DBConnect) . "</p>";
```



```
Successfully added the records.

Records: 10 Deleted: 0 Skipped: 0 Warnings: 0
```

Figure 9-18 Output of `mysqli_info()` function for a `LOAD DATA` query

Chapter Summary

- The `mysqli_connect()` function opens a connection to a MySQL database server.
- The `mysqli_close()` function closes a database connection.
- The `mysqli_select_db()` function selects a database.
- Writing code that anticipates and handles potential problems is often called bullet-proofing.
- The error control operator (@) suppresses error messages.
- The `die()` and `exit()` functions terminate script execution.
- The `mysqli_connect_errno()` function returns the error code from the last database connection attempt or zero if no error occurred.
- The `mysqli_connect_error()` function returns the error message from the last database connection attempt or an empty string if no error occurred.
- The `mysqli_errno()` function returns the error code from the last attempted MySQL function call or zero if no error occurred.
- The `mysqli_error()` function returns the error message from the last attempted MySQL function call or an empty string if no error occurred.
- The `mysqli_query()` function sends SQL statements to MySQL.
- A result pointer is a special type of variable that refers to the currently selected row in a resultset.
- The `mysqli_fetch_row()` function returns the fields in the current row of a resultset into an indexed array and moves the result pointer to the next row.
- The `mysqli_fetch_assoc()` function returns the fields in the current row of a resultset into an associative array and moves the result pointer to the next row.

- The `mysqli_num_rows()` function returns the number of rows in a query result, and the `mysqli_num_fields()` function returns the number of fields in a query result.
- The `mysqli_free_result()` function closes a resultset.
- You use the `CREATE DATABASE` statement with the `mysqli_query()` function to create a new database.
- You use the `DROP DATABASE` statement with the `mysqli_query()` function to delete a database.
- You use the `CREATE TABLE` statement with the `mysqli_query()` function to create a table.
- You use the `DROP TABLE` statement with the `mysqli_query()` function to delete a table.
- To identify a field as a primary key in MySQL, you include the `PRIMARY KEY` keywords when you first define a field with the `CREATE TABLE` statement. The `AUTO_INCREMENT` keyword is often used with a primary key to generate a unique ID for each new row in a table.
- You use the `INSERT` and `VALUES` keywords with the `mysqli_query()` function to add records to a table.
- You use the `LOAD DATA` statement and the `mysqli_query()` function with a local text file to add multiple records to a database.
- You use the `UPDATE`, `SET`, and `WHERE` keywords with the `mysqli_query()` function to update records in a table.
- You use the `DELETE` and `WHERE` keywords with the `mysqli_query()` function to delete records in a table.
- With queries that return results, such as `SELECT` queries, you can use the `mysqli_num_rows()` function to find the number of records returned from the query.
- The `mysqli_info()` function returns the number of operations for various types of actions, depending on the type of query.

Review Questions

1. MySQL support is enabled in PHP by default. True or False?
2. Which of the following functions closes a database connection?
   a. `close()`
   b. `mysqli_close()`
   c. `mysqli_free()`
   d. `mysqli_free_connect()`

3. To which of the following functions do you need to pass a variable representing the database connection? (Choose all that apply.)
   a. `mysqli_get_client_info()`
   b. `mysqli_get_host_info()`
   c. `mysqli_get_proto_info()`
   d. `mysqli_get_server_info()`
4. What is the correct syntax for selecting a database with the `mysqli_select_db()` function?
   a. `mysqli_select_db(`*`connection`*`)`
   b. `mysqli_select_db(`*`database`*`)`
   c. `mysqli_select_db(`*`connection`*`, `*`database`*`)`
   d. *`database`*` = mysqli_select_db(`*`connection`*`)`
5. Explain the types of errors that can occur when accessing MySQL databases and other types of data sources with PHP.
6. The following code structure prevents error messages from printing in the event that the database connection is not available. True or False?

```
$DBConnect = mysqli_connect("localhost", "dongosselin",
     "rosebud", "flightlog");
if (!$DBConnect)
      echo "<p>The database server is not available.</p>";
else  {
     echo "<p>Successfully connected to the database server.</p>";
     mysqli_close($DBConnect);
}
```

7. Explain the concept of bulletproofing your code.
8. Which of the following characters suppresses error messages in PHP?
   a. `*`
   b. `&`
   c. `#`
   d. `@`
9. Which of the following functions terminate script execution? (Choose all that apply.)
   a. `exit()`
   b. `bye()`
   c. `die()`
   d. `quit()`

10. Which of the following functions reports the error message from the last failed database connection attempt?
    a. `mysqli_connect_errno()`
    b. `mysqli_connect_error()`
    c. `mysqli_errno()`
    d. `mysqli_error()`
11. Explain what a result pointer is and how to create and use one.
12. Which of the following functions returns the fields in the current row of a resultset into an indexed array?
    a. `mysqli_data_fetch()`
    b. `mysqli_data_seek()`
    c. `mysqli_index_row()`
    d. `mysqli_fetch_row()`
13. Which of the following functions returns the fields in the current row of a resultset into an associative array?
    a. `mysqli_assoc_fetch()`
    b. `mysqli_fetch_keys()`
    c. `mysqli_fetch_assoc()`
    d. `mysqli_fetch_index()`
14. Write a simple code segment that demonstrates how to use the `mysqli_num_rows()` and `mysqli_num_fields()` functions to determine whether a SQL query returned results.
15. Which of the following functions closes a resultset to ensure that it doesn't keep taking up space in your computer's memory?
    a. `mysqli_free_result()`
    b. `mysqli_result_close()`
    c. `mysqli_free()`
    d. `mysqli_close_result()`
16. Write a simple code segment that demonstrates how to use the `mysqli_db_select()` function to check whether a database exists before you create or delete it.
17. Write a simple code segment that demonstrates how to use a `mysqli_query()` function to prevent your code from attempting to create a table that already exists.

18. Which of the following SQL keywords creates an autoincrementing field?
    a. `AUTO`
    b. `INCREMENT`
    c. `AUTO_INCREMENT`
    d. `AUTOINCREMENT`
19. Which of the following functions returns the number of rows affected by queries that do not return results, such as `INSERT`, `UPDATE`, and `DELETE` queries?
    a. `mysqli_affected_rows()`
    b. `mysqli_rows()`
    c. `mysqli_get_changed()`
    d. `mysqli_fetch_rows()`
20. The ________________ function returns the number of operations for various types of actions, depending on the type of query.
    a. `mysqli_get_info()`
    b. `mysqli_operations()`
    c. `mysqli_info()`
    d. `mysqli_fetch_actions()`

HANDS-ON PROJECTS

Hands-On Project 9-1

In this project, you create a hit counter script that keeps track of the number of hits a Web page receives. The number of hits will be stored as autoincrementing primary keys in MySQL.

1. Create a new document in your text editor and type the `<!DOCTYPE>` declaration, `<html>` element, document head, and `<body>` element. Use the strict DTD and "Hit Counter" as the content of the `<title>` element.
2. Add the following script section to the document body:

```
<?php
?>
```

3. Add the following statement to the script section to connect to the database. Replace *user* and *password* with the MySQL username and password you created in Chapter 8.

```
$DBConnect = @mysqli_connect("localhost", "user", "password")
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
```

4. Add the following statements to the end of the script section to create a database named `hit_counter` if it does not already exist:

```
$DBName = "hit_counter";
if (!@mysqli_select_db($DBConnect, $DBName)) {
     $SQLstring = "CREATE DATABASE $DBName";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
           Or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>You are the first visitor!</p>";
     mysqli_select_db($DBConnect, $DBName);
}
```

5. Add the following statements to the end of the script section to create a table named `count` if it does not already exist. The table consists of a single autoincrementing primary key field named `countID`.

```
$TableName = "count";
$SQLstring = "SELECT * FROM $TableName";
$QueryResult = @mysqli_query($DBConnect, $SQLstring);
if (!$QueryResult) {
     $SQLstring = "CREATE TABLE $TableName (countID SMALLINT NOT
NULL AUTO_INCREMENT PRIMARY KEY)";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
           Or die("<p>Unable to create the table.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
}
```

6. Add the following statements to the end of the script section to add a new row to the count table, which increments the `countID` field by one:

```
$SQLstring = "INSERT INTO $TableName VALUES(NULL)";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
           . "<p>Error code " . mysqli_errno($DBConnect)
           . ": " . mysqli_error($DBConnect)) . "</p>";
```

7. Finally, add the following statements to the end of the script section. The first statement uses the `mysqli_insert_id()` function to return the last value assigned to the `countID` field and the `echo()` statement prints the number of hits. The last statement closes the database connection.

```
$Hits = mysqli_insert_id($DBConnect);
echo "<h1>There have been $Hits hits to this page!</h1>";
mysqli_close($DBConnect);
```

8. Save the document as **HitCounter.php** in the Projects directory for Chapter 9.

9. Open **HitCounter.php** file in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Projects/HitCounter.php**. The first time you open the Web page, you should see the message about being the first visitor to the Web site, along with a hit count of 1. Reload the Web page a few times to see if the count increases.
10. Close your Web browser window.

Hands-On Project 9-2

In this project, you create a Web page that allows visitors to your site to sign a guest book that is saved to a database.

1. Create a new document in your text editor and type the `<!DOCTYPE>` declaration, `<html>` element, document head, and `<body>` element. Use the strict DTD and "Guest Book" as the content of the `<title>` element.
2. Add the following text and elements to the document body:

```
<h2>Enter your name to sign our guest book</h2>
<form method="get" action="SignGuestBook.php">
<p>First Name <input type="text" name="first_name" /></p>
<p>Last Name <input type="text" name="last_name" /></p>
<p><input type="submit" value="Submit" /></p>
</form>
```

3. Save the document as **GuestBook.html** in the Projects directory for Chapter 9.
4. Create a new document in your text editor and type the `<!DOCTYPE>` declaration, `<html>` element, document head, and `<body>` element. Use the strict DTD and "Guest Book" as the content of the `<title>` element.
5. Add the following script section to the document body:

```
<?php
?>
```

6. Add the following statements to the script section to ensure that visitors enter their first and last names:

```
if (empty($_GET['first_name']) || empty($_GET['last_name']))
     die("<p>You must enter your first and last name! Click your
browser's Back button to return to the Guest Book form.</p>");
```

7. Add the following statement to the script section to connect to the database. Replace *user* and *password* with the MySQL username and password you created in Chapter 8.

```
$DBConnect = @mysqli_connect("localhost", "user", "password")
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
```

8. Add the following statements to the end of the script section to create a database named `hit_counter` if it does not already exist:

```
$DBName = "guestbook";
if (!@mysqli_select_db($DBConnect, $DBName)) {
     $SQLstring = "CREATE DATABASE $DBName";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
          Or die("<p>Unable to execute the query.</p>"
          . "<p>Error code " . mysqli_errno($DBConnect)
          . ": " . mysqli_error($DBConnect)) . "</p>";
     echo "<p>You are the first visitor!</p>";
     mysqli_select_db($DBConnect, $DBName);
}
```

9. Add the following statements to the end of the script section to create a table named `count` if it does not already exist. The table consists of a single autoincrementing primary key field named `countID`.

```
$TableName = "visitors";
$SQLstring = "SELECT * FROM $TableName";
$QueryResult = @mysqli_query($DBConnect, $SQLstring);
if (!$QueryResult) {
     $SQLstring = "CREATE TABLE $TableName (countID SMALLINT
     NOT NULL AUTO_INCREMENT PRIMARY KEY,
     last_name VARCHAR(40), first_name VARCHAR(40))";
     $QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to create the table.</p>"
     . "<p>Error code " . mysqli_errno($DBConnect)
     . ": " . mysqli_error($DBConnect)) . "</p>";
}
```

10. Finally, add the following statements to the end of the script section. These `mysqli_query()` statements add the visitor to the database and the last statement closes the database connection.

```
$LastName = addslashes($_GET['last_name']);
$FirstName = addslashes($_GET['first_name']);
$SQLstring = "INSERT INTO $TableName VALUES(NULL, '$LastName',
     '$FirstName')";
$QueryResult = @mysqli_query($DBConnect, $SQLstring)
     Or die("<p>Unable to execute the query.</p>"
          . "<p>Error code " . mysqli_errno($DBConnect)
          . ": " . mysqli_error($DBConnect)) . "</p>";
echo "<h1>Thank you for signing our guest book!</h1>";
mysqli_close($DBConnect);
```

11. Save the document as **SignGuestBook.php** in the Projects directory for Chapter 9.
12. Open **GuestBook.html** file in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Projects/GuestBook.html**. Test the form to see if you can add your name to the database.
13. Close your Web browser window.

Hands-On Project 9-3

In this project, add a document to the Guest Book program you created in Hands-On Project 9-2. This document displays the entries in the guest book.

1. Create a new document in your text editor and type the `<!DOCTYPE>` declaration, `<html>` element, document head, and `<body>` element. Use the strict DTD and "Guest Book" as the content of the `<title>` element.
2. Add the following script section to the document body:

```
<?php
?>
```

3. Add the following statement to the script section to connect to the database. Replace *user* and *password* with the MySQL username and password you created in Chapter 8.

```
$DBConnect = @mysqli_connect("localhost", "user", "password")
     Or die("<p>Unable to connect to the database server.</p>"
     . "<p>Error code " . mysqli_connect_errno()
     . ": " . mysqli_connect_error()) . "</p>";
```

4. Add the following statements to the end of the script section to connect to the `guestbook` database. If the database does not exist, a message prints that the guest book does not contain any entries.

```
$DBName = "guestbook";
if (!@mysqli_select_db($DBConnect, $DBName))
     die("<p>There are no entries in the guest book!</p>");
```

5. Add the following statements to the end of the script section to select all the records in the `visitors` table. If no records are returned, a message prints that the guest book does not contain any entries.

```
$TableName = "visitors";
$SQLstring = "SELECT * FROM $TableName";
$QueryResult = @mysqli_query($DBConnect, $SQLstring);
if (mysqli_num_rows($QueryResult) == 0)
     die("<p>There are no entries in the guest book!</p>");
```

6. Add the following statements to the end of the script section to print the records returned from the `visitors` table:

```
echo "<p>The following visitors have signed our guest book:</p>";
echo "<table width='100%' border='1'>";
echo "<tr><th>First Name</th><th>Last Name</th></tr>";
$Row = mysqli_fetch_assoc($QueryResult);
do {
     echo "<tr><td>{$Row['first_name']}</td>";
     echo "<td>{$Row['last_name']}</td></tr>";
     $Row = mysqli_fetch_assoc($QueryResult);
} while ($Row);
```

7. Add the following statements to the end of the script section to close the database connection and the result pointer:

```
mysqli_free_result($QueryResult);
mysqli_close($DBConnect);
```

8. Save the document as **ShowGuestBook.php** in the Projects directory for Chapter 9.
9. Return to the **GuestBook.html** document in your text editor and add the following text and elements to the end of the document body:

```
<p><a href="ShowGuestBook.php">Show Guest Book</a></p>
```

10. Save the **GuestBook.html** file, and then open it in your Web browser by entering the following URL: **http://localhost/PHP_Projects/Chapter.09/Projects/GuestBook.html**. Click the Show Guest Book link to see if the script functions correctly.
11. Close your Web browser window.

Case Projects

In Chapter 6, you created versions of the following projects that saved data to text files. Create new versions of each project that store data in MySQL databases instead of text files. Save the documents you create for the following projects in the Cases directory for Chapter 9.

Case Project 9-1

Create a document with a form that registers users for a professional conference.

Case Project 9-2

Create a telephone directory application that saves entries to a single text file. You should include standard telephone directory fields in the database, such as first name, last name, address, city, state, zip, telephone number, and so on. Create a document as a main "directory," where you can select and retrieve records. Also, create one document that you can use to add new entries to the database and another document that you can use to edit entries.

Case Project 9-3

Create a Web page to be used for storing software development bug reports in a MySQL database. Include fields such as product name and version, type of hardware, operating system, frequency of occurrence, and proposed solutions. Include links on the main page that allow you to create a new bug report and update an existing bug report.

Case Project 9-4

Create a Web site for tracking, documenting, and managing the process of interviewing candidates for professional positions. On the main page, include a form with fields for the interviewer's name, position, and date of interview. Also include fields for entering the candidate's name, communication abilities, professional appearance, computer skills, business knowledge, and interviewer's comments. Clicking the Submit button should save the data in a MySQL database. Include a link for opening a document that displays each candidate's interview information.

Case Project 9-5

Create a Web page that stores airline surveys in a MySQL database. Include fields for the date and time of the flight, flight number, and so on. Also, include groups of radio buttons that allow the user to rate the airline on the following criteria:

- Friendliness of customer staff
- Space for luggage storage
- Comfort of seating
- Cleanliness of aircraft
- Noise level of aircraft

The radio buttons for each question should consist of the following options: No Opinion, Poor, Fair, Good, or Excellent. Separate text files should store the results of a single survey. Include a View Past Survey Results button on the main survey page that displays a list of past survey results.