17

Connecting to MySQL from a PHP Application

Throughout the book, you have learned about many different aspects of MySQL, including how to install MySQL, create databases and tables, and retrieve and manipulate data in those tables. You have even learned how to perform administrative tasks such as granting privileges to user accounts and optimizing your queries. As you learned to perform these tasks, you often worked with MySQL interactively by issuing SQL statements and commands from within the mysql client utility. The majority of access to MySQL databases is through applications that use application programming interfaces (APIs) to connect to MySQL and issue SQL statements. As a result, the applications — along with the APIs — are the primary vehicles available to users to interact with MySQL data. At the very least, the applications allow users to view data that is retrieved from the database, but in many cases, they are also able to add to, update, or delete that data.

Because of the importance that applications play in accessing data in a MySQL database, the final three chapters of the book focus on connecting to a MySQL database from an application and then accessing data in that database. This chapter explains how to work with a MySQL database from a PHP application. Chapter 18 focuses on Java applications, and Chapter 19 focuses on ASP.NET applications. In each chapter, you learn how to create a database connection, retrieve data, and then modify the data, all in that particular application language. The book begins with PHP because it is one of the most common application languages used to connect to MySQL. The PHP/MySQL application has been implemented worldwide for systems that vary greatly in size and number of users. This chapter, then, provides you with the information you need to allow your PHP application to communicate with MySQL. Specifically, the chapter does the following:

- Introduces you to PHP and how it communicates with a MySQL server and its databases
- Explains how to build a PHP application that connects to a MySQL database, retrieves data from that database, inserts data in the database, modifies that data, and then deletes the data

Introduction to PHP

PHP is a server-side scripting language that is used in Web-based applications. PHP also refers to the preprocessor that is installed on your system when you install PHP. (PHP stands for PHP: Hypertext Preprocessor, which makes PHP a recursive acronym.) The PHP scripts are embedded in the Web pages, along with HTML, to create a robust Web application that can generate dynamic content that is displayed to the user. PHP is similar in some ways to other application languages. For example, PHP, Java, and C# (one of the languages used in ASP.NET applications) are all considered object-oriented procedural languages and are derived from a similar source — the C and C++ programming languages. PHP is used primarily for Web-based applications, whereas Java and C# are used not only to build Web-based applications but also to build larger PC-based and client/server applications that require a more rigid structure.

Although PHP is not as robust and as extensive as other languages, it does have its place, particularly for less complex Web-based applications. PHP is usually easier to use and implement, and the structure tends to be more flexible than those of some of the more complex languages. For example, you do not need to declare variables in PHP — you can simply start using them — and there is more flexibility in terms of the values that you can assign to them, compared to the other languages.

Another advantage to using PHP is that, like MySQL, it is an open-source technology. This means that you can download PHP for free and start experimenting with it immediately. By using Linux, MySQL, PHP, and Apache (an open-source Web server) in combination with one another, you can set up a completely open-source environment for your Web applications.

Like any regular HTML Web page, the pages in a PHP application are hosted by a Web server, one that supports PHP applications. Apache is such a Web server (and probably the most commonly used Web server for PHP applications). The PHP application connects with MySQL through the MySQL PHP API, which is included in PHP when you install it on your computer. As a result, in order to run a PHP Web-based application, you must have both PHP and a Web server installed. You can install MySQL on the same computer or on a different computer, depending on your environment and your security requirements and on how you will be using the PHP configuration. For example, if you are developing a PHP application, you might want to install everything on one computer; however, in a production environment, you may want to install MySQL on a back-end computer, perhaps separated from the front-end computer by a firewall.

When a Web browser on a client computer connects to a PHP page, the PHP script is executed and the results are returned to the browser, along with any HTML on that page. As you learn later in the chapter, the PHP script is separated from the HTML by special opening and closing tags that are specific to PHP. When the PHP preprocessor sees the opening tag, it switches from HTML mode to PHP mode. When it sees the closing tag, it switches back to HTML mode. Everything between the opening tag and closing tag is treated as PHP script.

The key, then, in connecting to a MySQL database and accessing data from a PHP application is to include in your application the PHP script necessary to communicate with the MySQL database. To this end, this chapter covers the information you need to connect to the database, retrieve data, and modify that data. Although the chapter does not provide an in-depth look at PHP, it does provide the basic details necessary to create a data-driven application. The chapter does assume, though, that you are already familiar with Web-based development, HTML, and, to some extent, PHP. From this foundation, you should be able to use the information in this chapter to build a PHP/MySQL data-driven application.

This chapter was written based on the following configuration: Apache 2.0.52, PHP 5.0.2.2, and MySQL 4.1.6 installed on a Windows XP computer. Details about the configuration of your operating system, Web server, and PHP are beyond the scope of the book. Be sure to view the necessary product documentation when setting up your application environment. For more information about Apache, go to `www.apache.org`*. For more information about PHP, go to* `www.php.net`*.*

Building a Data-Driven PHP Application

When you create a PHP application that accesses data in a MySQL database, you must first establish a connection to the database from that application. Once the connection has been established, you can then use PHP script elements, along with SQL statements, to retrieve or modify data. The remaining part of the chapter describes each aspect of database access. First you learn how to connect to a database from your PHP application, and then you learn how to retrieve data from the database. Next you learn how to insert data, modify data, and delete data.

The examples in these chapters focus on the PHP scripts used to create a basic Web-based application. The principles behind connecting to a database and accessing data, however, can apply to any Web or non-Web application written in other programming languages.

Connecting to a MySQL Database

The first step to working with a database in a PHP application is to create a database connection. Creating that connection consists of two components: connecting to the server and choosing a database. You connect to the server by using the `mysql_connect()` function. The function is a PHP function (specific to MySQL) that facilitates the process of authenticating the user and establishing a connection to the server. If the MySQL server authenticates the user, a link is established between the application and the MySQL server. This link is referred to as a resource link.

PHP supports numerous functions specific to MySQL. As you become more familiar with interacting with MySQL through your PHP applications, you'll discover that much of that interaction is through those functions. All PHP functions related specifically to MySQL begin with "mysql."

The `mysql_connect()` function takes three parameters: the hostname of the system where the MySQL server resides, the user account name that is authorized to connect to the MySQL server, and a password for that account. If no parameters are specified with the `mysql_connect()` function, PHP assumes that the hostname is localhost, the username is the user who owns the current server process, and the password is blank. In general, it's normally a good idea to specify all three parameters so that you always know exactly how the connection is being established.

Once that resource link has been defined, you should use the `mysql_select_db()` function to select the database that contains the data to be accessed. All queries to the MySQL server are then directed to that database, via the resource link. When using the `mysql_select_db()` function, you must specify the database name as the first parameter, and optionally you can specify a link identifier as a second parameter. A link identifier is a merely a way to reference the resource link, which is usually done through a variable that has been assigned a value based on the `mysql_connect()` function. If no link identifier is specified, PHP assumes that the last opened link should be used. If no link is open, PHP issues a `mysql_connect()` function that includes no parameters.

Generally, the easiest way to set up your MySQL connection is to define a variable that takes the `mysql_connect()` function as a value and then use that variable in your `mysql_select_db()` function. For example, suppose that you want to create a connection to the local computer for the cduser account. You could define a variable similar to the following:

```
$link = mysql_connect("localhost", "cduser", "pw")
   or die("Could not connect: " . mysql_error());
```

The name of the variable is `$link`. In PHP, all variable names begin with a dollar sign ($). To assign a value to a variable, you specify the variable name, followed by an equal sign, and then followed by the variable value. Variables are used to store various data including data retrieved from the database. Unlike some of the other programming languages, in PHP you do not have to declare the variable before assigning a value to it. By simply assigning a value to the variable, the variable is automatically available for use in your code.

In the preceding example, the variable is assigned a value based on the `mysql_connect()` function, which takes three parameters. The first parameter, `localhost`, refers to the computer where the MySQL server resides. The second parameter, `cduser`, is the name of the user account in MySQL that should be used by the application. Finally, the third parameter, `pw`, is the password that should be used for the user account. Notice that each string parameter is enclosed in double quotes and separated by a comma. You must separate parameters with a comma in a PHP function call.

The `mysql_connect()` function initializes the connection to the MySQL server and returns a link reference that is assigned to the `$link` variable. Because the link reference is assigned to the `$link` variable, you can reference the variable in your code whenever you want to invoke that particular link reference.

The second line of code in the previous example defines an `or die` statement that is executed if the `mysql_connect()` function fails (in other words, if a connection cannot be established). The `or die` condition uses the `or` operator and the `die()` function to specify the response that should be returned if a connection cannot be established. The `die()` function takes one parameter, which is the message to be returned. In this case, that message is made up of two parts: "Could not connect:" and the `mysql_error()` function, which returns the MySQL error that is received if the connection fails. Notice that the two parts are connected by a period, which indicates that they should be concatenated. In other words. "Could not connect:" should be followed by the error message — all in one response.

> *Notice that the entire two lines of code are terminated by a semi-colon (;). In PHP, each complete command ends with the semi-colon. If the code spans multiple lines, the semi-colon appears only at the end of the last line, just as it does when you're executing SQL statements in the mysql client utility.*

Once you have defined a variable to initialize a connection, you can use the variable as an argument in the `mysql_select_db()` function to connect to a specific server and database. For example, in the following line of code, the `mysql_select_db()` function takes two parameters: the name of the database and the `$link` variable:

```
mysql_select_db("CDDB", $link);
```

The `mysql_select_db()` function selects the database to be accessed by the application. The first parameter in the `mysql_select_db()` function is `CDDB`, which is the name of the database, and the second parameter is `$link`, which is the variable defined in the preceding example. Because the `$link` variable is assigned a link reference, that link reference is used to connect to the CDDB database. If the link reference is not included, the function uses the last operating resource link.

At the end of your PHP code, after you've processed all other statements related to the MySQL database, you should use the `mysql_close()` function to close your connection to the database. The function can take the resource link as a parameter so that you can close that connection. For example, the following code closes the connection established by the `$link` variable:

```
mysql_close($link);
```

If no parameter is specified, the function uses the last resource link that was established. Whether or not you specify an argument, you should always use the `mysql_close()` function when a database connection is no longer needed.

Retrieving Data from a MySQL Database

Once you have established your connection to the MySQL server and selected a database, you can retrieve data that can then be displayed by your PHP application. The primary mechanism that you use to retrieve data is the `SELECT` statement, written as it is written when you access data interactively. The `SELECT` statement alone, though, is not enough. You need additional mechanisms that pass that `SELECT` statement to the MySQL server and that then process the results returned by the statement.

When you're retrieving data from a MySQL database from your PHP application, you generally follow a specific set of steps:

1. Initialize a variable whose value is the `SELECT` statement.
2. Initialize a variable whose value is the result set returned by the `SELECT` statement. The variable uses the `mysql_query()` function to process the `SELECT` statement.
3. Use the second variable to process the results, normally in some sort of conditional structure.

Now take a look at this process step by step to get a better idea of how it works. For example, the first step is to define a variable to hold the `SELECT` statement. In the following code, a variable named `$selectSql` is defined:

```
$selectSql = "SELECT CDID, CDName, InStock, OnOrder, ".
             "Reserved, Department, Category from CDs";
```

Notice that the value assigned to the variable is a `SELECT` statement. If you want to split the statement into multiple lines, as has been done here, you enclose each portion of the statement line in double quotes. In addition, for each line other than the last line, you end the line with a period. This indicates that the two lines are to be concatenated when executed. Only the final line is terminated with a semi-colon.

> *Notice that a semi-colon follows the PHP statement, but not the SQL statement itself (inside the quotes). You do not include a semi-colon after the* `SELECT` *statement as you would when working with MySQL interactively in the mysql client utility.*

Putting the `SELECT` statement in a variable in this way makes it easier to work with the statement in other places in the code. For example, the next step in retrieving data is to create a variable that stores the results returned by the `SELECT` statement, as shown in the following example:

```
$result = mysql_query($selectSql, $link);
```

As you can see, the value of the `$result` variable is based on the `mysql_query()` function. The function sends the query to the active database for the specified link identifier. Whenever you use the `mysql_query()` function, the first parameter must be the SQL query, and the second parameter, which is optional, can be the resource link. If no resource link is specified, the most current link is used.

In the preceding example, variables are used for both parameters. The first variable, `$selectSql`, contains the `SELECT` statement that is to be sent to the database. The second variable, `$link`, contains the resource link that you saw defined in an earlier example. When the `mysql_query()` function processes the `SELECT` statement, the results are assigned to the `$result` variable, which can then be used in subsequent code to display results in the application.

After you've defined the `$result` variable, you can use an `if` statement to return an error message if the query results in an error, in which case the value of `$result` is set to false. In the following example, the `if` statement checks the value of `$result`, and then, if necessary, returns an error:

```
if (!$result)
  die("Invalid query: " . mysql_error(). "<br>".$selectSql);
```

The `if` statement first specifies the condition under which the `die()` function should be executed. The condition (which is based on the `$result` variable) is enclosed in parentheses. Notice that an exclamation point (!) precedes the variable name. The exclamation point means *not* and is used to indicate that, if the query (as represented by the variable) is not true, then the `die()` function should be executed. If the `die()` function is executed, it returns the message "Invalid query:" followed by the error returned by MySQL. The next part of the parameter — `<br>` — indicates that a line break should be inserted and the rest of the message should start on a new line. The `<br>` symbol is HTML code. You can insert HTML in certain PHP command as long as you enclose the HTML in double quotes. Because of the `<br>` code, the original `SELECT` statement (as displayed by the `$selectSql` variable) is then displayed on a line beneath the first part of the message returned by the `die()` function.

A query that executes successfully but that returns no data is treated as a successfully executed statement and does not generate an error. This means that the `$result` *variable still evaluates to true, but no results are displayed on the Web page.*

Processing the Result Set

Once you have a result set to work with, you must process the rows so that they can be displayed in a usable format. In PHP, you must set up your code to be able to process one row at a time. You can do this by setting up a loop condition and by using a function that supports the row-by-row processing. For example, you can use the `while` command to set up the loop and the `mysql_fetch_array()` function to process the rows, as shown in the following example:

```
while($row = mysql_fetch_array($result))
{
   $cdId = $row["CDID"];
   $cdName = $row["CDName"];
   $inStock = $row["InStock"];
   $onOrder = $row["OnOrder"];
   $reserved = $row["Reserved"];
   $department = $row["Department"];
   $category = $row["Category"];
   printf("$cdId, $cdName, $inStock, $onOrder, $reserved, $department, $category");
}
```

The `while` command tells PHP to continue to execute the block of commands as long as the current condition is not false, meaning that `$row` has a value. The condition in this case is defined by `$row = mysql_fetch_array($result)`. The `mysql_fetch_array()` function returns the values from the query's result set one row at a time. Consequently, the `while` loop is executed for each row that is returned by the `mysql_fetch_array()` function. When the function no longer returns a row, it returns a value of false, and the `while` command stops executing. For example, if a query's result set contains 15 rows, the `while` command is executed 15 times.

The current row returned by the `mysql_fetch_array()` function is placed in the `$row` variable, which is an array of the columns. Each time the `while` command is executed, a new row is inserted in the variable. The variable can be used to return specific values as part of the loop construction. For example, as long as the `while` condition is not false, the array holds the current row. As a result, each column of the `SELECT` statement can be retrieved from `$row` by name. For instance, the `$cdId = $row["CDID"]` command retrieves the value that was returned from the CDID column and places it in the `$cdId` variable. Each time the `$row` variable is updated (when the `mysql_fetch_array()` function returns the next row), the `$cdId` variable is updated with the current CDID value.

To illustrate this, suppose that the first row returned by your query contains the following values:

- CDID = 101
- CDName = Bloodshot
- InStock = 10
- OnOrder = 5
- Reserved = 3
- Department = Popular
- Category = Rock

The `mysql_fetch_array()` function returns this row from the result set, which is available through the `$result` variable. Because the `$row` variable is defined to equal the `mysql_fetch_array()` function, the variable will contain the values included in that row. The `$row` variable and `mysql_fetch_array()` function are part of the `while` construction, however, so the value of the `$row` variable is used as part of the `while` loop. For instance, the current CDID value in the `$row` variable is 101. As a result, the `$cdId` variable is assigned a value of 101. For each column returned by the `mysql_fetch_array()` function, the current value is assigned to the applicable variable. The column-related variables can then be used to display the actual values returned by the `SELECT` statement.

The next step in the loop construction is to use the PHP `printf()` function to display the values saved to the column-related variables for that specific row. Each time the `while` command is executed, the values for a specific row (taken from the variables) are displayed on your Web page. By the time the `while` command has looped through the entire result set, all rows returned by the query are displayed. (Later, in the Try It Out sections, you see the results of using the `while` command to create a loop construction.)

Manipulating Data in PHP

When you retrieve data from a database, you might find that you want to manipulate or compare data in some way in order to display that data effectively. As a result, PHP includes several functions that support these types of operations. Two functions important to working with data are the `strlen()` function and the `strcmp()` function.

The `strlen()` function returns the length of a string. For example, if a string value is *book,* the `strlen()` function returns a value of 4.

The `strcmp()` function works differently from the `strlen()` function. Rather than checking the length of a string, it compares two strings to determine whether they have an equal value. The strings are compared based on a typical alphabetic sorting. For instance, banana is greater than orange. If the two strings are equal, the function returns a 0. If the first string is greater than the second string, the function returns a 1. If the first string is less than the second string, the function returns a -1. For example, if the first string is apple and the second string is strawberry, the function returns 1, but if the first string is apple and the second string is apple, the function returns 0.

Now take a look at how these two functions can be used in PHP script to manipulate data. Assume that your script includes a variable named `$department` and that a value has been assigned to that variable. You can use an `if` statement to specify conditions that use the two functions, as shown in the following example:

```
if((strlen($department) !=0) && (strcmp($department, "Classical")==0))
{
   printf("The $cdName CD is classified as $category.");
}
```

The `if` statement includes two conditions. The first condition uses the `strlen()` function to determine whether the length of the string stored in the `$department` variable is equal to an amount other than 0. The exclamation point/equal sign (`!=`) operator means *does not equal.* In other words, the function determines whether the variable is not empty. If it is not (it has a value that is not 0), then the condition evaluates to true.

The `strcmp()` function compares the value held in the `$department` variable to the string Classical. If `$department` equals Classical, then a 0 is returned. The `if` condition specifies that the two strings must be equal (must return a value of zero). The double equal signs (`==`) operator means *is equal* and is followed by a 0. In other words, the result returned by the `strcmp()` function must equal 0 in order for the condition to evaluate to true. The two conditions are then connected by the double ampersand (`&&`) operator, which means *and.* As a result, both conditions must evaluate to true for the `if` statement to be executed.

If both conditions evaluate to true, then the `printf()` function prints the message enclosed in the parentheses, replacing any variables with their actual value. For example, if the `$cdName` variable is currently set to Bloodshot, and the `$category` variable is currently set to Rock, the message is printed as "The Bloodshot CD is classified as Rock."

Another useful function when working with data is `substr()`, which extracts a specified number of characters from a string. The function takes three parameters. The first of these is the actual string from which you want to extract a value. The second parameter is the starting position from where you begin to extract characters. The function begins extracting values at the identified position. (The first position is 0.) The third argument specifies the number of characters to extract. For example, the function `substr("house", 2, 3)` returns the value *use*. If you want to extract a value starting with the first character, you should specify 0 as your starting position.

Now take a look at an example to better explain how the `substr()` function works. First, assume that the following three variables have been defined:

```
$firstName = "Johann";
$middleName = "Sebastian";
$lastName = "Bach";
```

The variables can then be used along with the `substr()` function to concatenate the composer's name, as shown in the following example:

```
$abbreviatedName = substr($firstName, 0, 1) . ". " .
                   substr($middleName, 0, 1) . ". " . $lastName;

printf($abbreviatedName);
```

In this example, a variable named `$abbreviatedName` is being defined. The variable concatenates the three variables by connecting them with periods, which indicates to PHP that these lines should be treated as a single unit. Note, however, that a period and space are enclosed in quotes. Because they're in quotes, they're treated as string values and so are part of the actual value.

Now take a look at the first instance of the `substr()` function. Notice that it takes the `$firstName` variable as the first argument. The second argument, which is 0, indicates that the substring should be created beginning with the first letter, and the third parameter, 1, indicates that only one character should be used. As a result, the first `substr()` function returns a value of *J.* The second instance of the `substr()` function works the same way and returns a value of *S.* When all the pieces are put together, the `$abbreviatedName` variable is set to the value *J. S. Bach*, which is then printed on the Web page through the use of the `printf()` function.

Converting Date Values

Often, when you're retrieving a date value from a MySQL database, you want to convert the value so that it can be displayed in a more readable format. PHP provides several date-related and time-related functions that allow you to work with date/time values. Two functions that are particularly useful are the `date()` and `strtotime()` functions. The `date()` function extracts part of a date from a value and puts it in a specified format. The `strtotime()` function converts a date that is retrieved as a string value (such as 2004-10-31) and converts it in a numerical format (as a timestamp) that can then be used by the `date()` function. For example, the following statement sets the `$saleDatePrint` variable to a date based on a value in the `$saleDate` variable:

```
$saleDatePrint = date("m-d-Y", strtotime($saleDate));
```

When a date is retrieved from MySQL, it is retrieved as a string value, displayed with the year first, then the month, and then the day. In this example, that value is stored in the `$saleDate` variable. The `strtotime()` function converts that value to a numerical timestamp value, which is then used by the `date()` function. The `date()` function takes two parameters. The first parameter defines the format that should be used. In this case, the format is `m-d-Y`, which translates to *<month>-<day>-<four-digit year>*. The second parameter is the actual date value. Because the `strtotime()` function is used, the `$saleDate` value is converted to the numerical timestamp. The `date()` function then extracts the month, day, and year from the timestamp and displays it in the desired format.

Working with HTML Forms

When working with Web-related applications, you often need to pass data from the client browser to the server. This is often done through the use of a *form*, which is an element on an HTML page that allows a user to submit data that can be passed to the server. For example, a form can be used to allow a user to log on to the application. The user can submit an account name and password, which the form then sends to a specified location.

An HTML form supports two types of posting methods: `POST` and `GET`. The `POST` method sends the data through an HTTP header. An HTTP header is a mechanism that allows commands and data to be sent to and from the browser and server. The `GET` method adds the data to a URL. Because the data is added to the URL, it is visible when the URL is displayed. For this reason, using the `POST` method is often preferable because the posted data is hidden.

> *A thorough discussion of HTML forms and HTTP headers is beyond the scope of this book; however, there are plenty of resources that describe forms, headers, and all other aspects of HTML and HTTP extensively. If you're not familiar with how to work with forms and headers, be sure to read the appropriate documentation.*

Once the data has been posted by the form, you can use elements in PHP to retrieve the data. To support form functionality, PHP provides the `$_POST` array and the `$_GET` array. An array is a set of data values that are stored as a unit but can be referenced as individual values. Data posted by a form is available in PHP through the array. The `$_POST` array provides the data submitted by a form if that form uses the `POST` method. The `$_GET` array provides the data submitted by a form if that form uses the `GET` method. Each value in the array is referenced by the name that is used on the form. For example, suppose that you create a form on your HTML page that includes the following `<input>` element:

```
<input type="text" name="department">
```

The input element determines the type of action that the user takes. In this case, the input type shown here is text, which means that a text box is displayed on the Web page and the user can enter a value in the text box. That value is then submitted to the target file either through the HTTP header or as an add-on to the URL. Your PHP script can then use that value by accessing the `$_POST` array and the `$_GET` array.

When you use one of the arrays to access a particular value, you must reference the name that is assigned to the input element. In the case of the previous example, the value that the user enters is referenced by using the name department, which is the name assigned to the input element. For example, suppose that the input element shown previously is part of a form that has been defined with the `POST` method. You can then use the `$_POST` array to retrieve this value that was entered by the user in the form, as shown in the following PHP script:

```
if(isset($_POST["department"]))
   $department = $_POST["department"];
```

The script is an `if` statement that uses both the `$_POST` array and the `isset()` function. The `isset()` function tests whether a variable has a value. The function can also be used to check if an array position holds a value, as in the preceding example. If the value has been assigned to the variable, the `if` condition evaluates to true, and the next line of code is executed. In this case, the `isset()` function is used to determine whether the `$_POST` array contains a value for the department input element. If a value exists,

that value is assigned to the `$department` variable. As you can see, the `$_POST` array allows you to retrieve the data that was posted by the user and use that data in your PHP code.

Redirecting Browsers

Web application pages can sometimes decide that the user should be redirected to another page. This means that the current page stops processing and the server loads a new page and processes it. This process is usually part of a condition that specifies if a certain result is received; then an action must be taken based on that result. For example, suppose that you want to redirect a user if that user enters Classical in the department `<input>` element. You can set up your PHP code in a way similar to the following:

```
if(strcmp($_POST["department"], "Classical") == 0)
{
   header("Location: ClassicalSpecials.php");
   exit;
}
```

As you can see, a `strcmp()` function is used to compare the department value returned by the `$_POST` array to the string Classical. If the comparison evaluates to 0 (the values are equal), the `header()` function and the `exit` command are executed. The `header()` function redirects the client browser to the specified page (in this case, ClassicalSpecials.php). You must include the `Location:` prefix when specifying the new page. The `exit` command terminates the execution of the current page.

> *Technically, the HTTP 1.1 specification says that you should use an absolute URL when providing the filename for the* `Location` *argument. Many clients take relative URLs, which is what we used here. If your application will be accessed by different types of browsers, you would probably want to use a complete URL such* `http://localhost/cdsales/ClassicalSpecials.php`.

Because you use the `header()` function and `exit` command in the context of an `if` statement, the user is redirected and the current page terminated only if the department value in the `$_POST` array equals `Classical`. Otherwise, the current page continues to be executed.

Working with Include Files

There might be times when you want to execute PHP script that is in a file separate from your current file. For example, suppose that your application includes PHP script that is repeated often. You can put that script in a file separate from your primary file and then call that file from the primary file. The second file is referred to as an include file, and you can simply reference it from your primary file to execute the script in the include file.

To reference an include file from your PHP script, you must use the `include` command, followed by the name of the include file. If the file is located someplace other than the local directory, you must also specify the path. For example, the following `if` statement uses the `include` command to execute the PHP script in the ClassicalSpecial.php file:

```
if (strcmp($_POST["department"], "Classical") == 0)
{
   include "ClassicalSpecials.php";
}
```

The `if` condition specifies that the department value in the `$_POST` array must equal Classical. If the condition is met, the include file is accessed and the script in the file is executed as though it were part of the current file. It would be the same as if the code in the include file actually existed in the primary file.

Now that you've been introduced to many of the basic PHP elements that you can use when retrieving and displaying data from a MySQL database, you're ready to build an application.

Creating a Basic PHP Application

In the Try It Out sections in this chapter, you build a simple Web application that allows you to view the transactions in the DVDRentals database. You are also able to add a transaction, edit that transaction, and then delete it from the database. As you'll recall when you designed the DVDRentals database, transactions are actually a subset of orders. Each order is made up of one or more transactions, and each transaction is associated with exactly one order. In addition, each transaction represents exactly one DVD rental. For example, if someone were to rent three DVDs at the same time, that rental would represent one order that includes three transactions. As a result, one row would be added to the Orders table to represent that order, and three rows would be added to the Transactions table to represent those orders. The rows in the Transactions table would then include a reference to the order ID.

The application that you build in this chapter is very basic and does not represent a complete application, in the sense that you would probably want your application to allow you to create new orders, add DVDs to the database, and add and edit other information. The purpose of this application is merely to demonstrate how you connect to a MySQL database from PHP, how you retrieve data, and how you manipulate data. The principles that you learn here can then be applied to any PHP application that must access data in a MySQL database.

When creating a Web application such as a PHP application, you usually find that you are actually programming in three or four different languages. For example, you might use PHP for the dynamic portions of your application, HTML for the static portions, SQL for data access and manipulation, and JavaScript to perform basic page-related functions. The application that you create in this chapter uses all four languages. At first this might seem somewhat confusing; however, the trick is to think about each piece separately. If you are new to these technologies, try doing each piece separately and then integrating the pieces. The application is fully integrated and can be run and examined to see how these technologies work. Keep in mind, however, that the focus of the Try It Out sections is to demonstrate PHP and SQL, so they do not contain detailed explanations about JavaScript and HTML. However, you cannot develop a PHP application without including some HTML, and JavaScript is commonly implemented in Web-based applications. As a result, in order to show you a realistic application, HTML and Java Script are included, but a discussion of these two technologies is well beyond the scope of this book. Fortunately, there are ample resources available for both of them, so be sure to consult the appropriate documentation if there is an HTML or JavaScript concept that you do not understand.

To support the application that you will create, you need two include files: one that contains HTML styles and one that contains the JavaScript necessary to support several page-related functions. You can download the files from `www.wrox.com`, or you can copy them from here. The first of these files is dvdstyle.css, which controls the HTML styles that define the look and feel of the application's Web pages. The styles control the formatting of various HTML attributes that can be applied to text and other objects. The following code shows the contents of the dvdstyle.css file.

```
table.title{background-color:#eeeeee}

td.title{background-color:#bed8e1;color:#1a056b;font-family:sans-serif;font-weight:
bold;font-size: 12pt}

td.heading{background-color:#486abc;color:#ffffff;font-family:sans-serif;font-
weight: bold;font-size: 9pt}

td.item{background-color:#99ff99;color:#486abc;font-family:sans-serif;font-weight:
normal;font-size: 8pt}

input.delete{background-color:#990000;color:#99ffff;font-family:sans-serif;font-
weight: normal;font-size: 8pt}

input.edit{background-color:#000099;color:#99ffff;font-family:sans-serif;font-
weight: normal;font-size: 8pt}

input.add{background-color:#000099;color:#99ffff;font-family:sans-serif;font-
weight: normal;font-size: 8pt}

td.error{background-color:#ff9999;color:#990000;font-family:sans-serif;font-weight:
bold;font-size: 9pt}
```

When you create an HTML element in your code, you can reference a particular style in the dvdstyle.css file, and then that style is applied. For example, the dvdstyle.css file includes the following style definition:

```
td.item{background-color:#99ff99;color:#486abc;font-family:sans-serif;font-weight:
normal;font-size: 8pt}
```

The `td.item` keywords identify the style definition. The td refers to the type of style definition, which in this case is a cell in a table, and item is the unique name given to this particular definition. The options defined in the paragraph are the various styles that apply to this definition. You can then reference this style definition in your HTML code. For example, if you are creating a table and you want a cell in that table to use this style, you would reference the *item* style name.

Whether you copy the file from the Web site or create the file yourself, you should save the file to the same directory where your PHP pages are stored. You can then modify the styles to meet your own needs.

The second file that you need to support the application is the dvdrentals.js file, which contains the JavaScript support functions for the Web form submission. These functions allow the program to manipulate the command values and the values of the form's action parameter. By using this technique, a user button click can redirect the form to a different page. The following code shows the contents of the dvdrentals.js file:

```
function doEdit(button, transactionId)
{
   button.form.transaction_id.value = transactionId;
   button.form.command.value = "edit";
   button.form.action = "edit.php";
```

```
    button.form.submit();
}

function doAdd(button)
{
    button.form.transaction_id.value = -1;
    button.form.command.value = "add";
    button.form.action = "edit.php";
    button.form.submit();
}

function doDelete(button, transactionId)
{
    var message = "Deleting this record will permanently remove it.\r\n" +
                  "Are you sure you want to proceed?";

    var proceed = confirm(message);

    if(proceed)
    {
        button.form.transaction_id.value = transactionId;
        button.form.command.value = "delete";
        button.form.submit();
    }
}

function doCancel(button)
{
    button.form.command.value = "view";
    button.form.action = "index.php";
    button.form.submit();
}

function doSave(button, command)
{
    button.form.command.value = command;
    button.form.submit();
}
```

The dvdrentals.js includes numerous function definitions. For example, the following JavaScript statement defines the `doEdit()` function:

```
function doEdit(button, transactionId)
{
    button.form.transaction_id.value = transactionId;
    button.form.command.value = "edit";
    button.form.action = "edit.php";
    button.form.submit();
}
```

The `doEdit()` function can be called from your HTML code, usually through an `<input>` element that uses a button click to initiate the function. The `doEdit()` function takes two parameters: `button` and

`transactionId`. The `button` parameter is used to pass the HTML button object, which references the form element in the JavaScript function, and the `transactionId` parameter holds the transaction ID for the current record. The `transactionId` value, along with a command value of `edit`, is submitted to the form in the edit.php file when that file is launched. Again, whether you copy the file from the Web site or create the file yourself, you should save the dvdrentals.js file to the same directory where your PHP pages are stored in your Web server.

Once you've ensured that the dvdstyle.css and dvdrentals.js files have been created and added to the appropriate directory, you're ready to begin creating your application. The first file that you create — index.php — provides the basic structure for the application. The file contains the PHP script necessary to establish the connection to the DVDRentals database, retrieve data from the database, and then display that data. The page lists all the transactions that currently exist in the DVDRentals database. In addition, the index.php file provides the foundation on which additional application functionality is built in later Try It Out sections. You can download any of the files used for the DVDRentals application created in this chapter at the Wrox Web site at `www.wrox.com`.

Try It Out — Creating the index.php File

The following steps describe how to create the index.php file, which establishes a connection to the DVDRentals database and retrieves transaction-related data:

1. The first part of the index.php file sets up the basic HTML elements that provide the structure for the rest of the page. This includes the page header, links to the dvdstyle.css and dvdrentals.js files, and the initial table structure in which to display the data retrieved from the DVDRentals database. Open a text editor, and enter the following code:

```
<html>
<head>
   <title>DVD - Listing</title>
   <link rel="stylesheet" href="dvdstyle.css" type="text/css">
   <script language="JavaScript" src="dvdrentals.js"></script>
</head>

<body>
<p></p>

<table cellSpacing="0" cellPadding="0" width="619" border="0">
<tr>
   <td>
      <table height="20" cellSpacing="0" cellPadding="0" width="619"
bgcolor="#bed8e1" border="0">
      <tr align=left>
         <td valign="bottom" width="400" class="title">
            DVD Transaction Listing
         </td>
      </tr>
      </table>
      <br>
      <table cellSpacing="2" cellPadding="2" width="619" border="0">
      <tr>
         <td width="250" class="heading">Order Number</td>
         <td width="250" class="heading">Customer</td>
```

```
            <td width="250" class="heading">DVDName</td>
            <td width="185" class="heading">DateOut</td>
            <td width="185" class="heading">DateDue</td>
            <td width="185" class="heading">DateIn</td>
        </tr>
```

2. The next section of the file creates the connection to the MySQL server and then selects the DVDRentals database. Add the following code after the code you added in Step 1:

```
<?
// Connect to server or return an error
$link = mysql_connect("localhost", "mysqlapp", "pw1")
   or die("Could not connect: " . mysql_error());

// Select database or return an error
mysql_select_db("DVDRentals", $link)
   or die("Unable to select database: . mysql_error()");
```

The user account specified in this section of code — mysqlapp — is an account that you created in Chapter 14. The account is set up to allow you to connect from the local computer. If you did not set up this account or plan to connect to a host other than the local host, you must create the correct account now. If you want to connect to the MySQL server with a hostname or username other than the ones shown here, be sure to enter the correct details. (For information about creating user accounts, see Chapter 14.)

3. In the following section you create the query that retrieves data from the DVDRentals database and stores the results of that query in a variable. Add the following code to your file:

```
// Construct the SQL statement
$selectSql = "SELECT ".
             "Transactions.TransID, ".
             "Transactions.OrderID, ".
             "Transactions.DVDID, ".
             "Transactions.DateOut, ".
             "Transactions.DateDue, ".
             "Transactions.DateIn, ".
             "Customers.CustFN, ".
             "Customers.CustLN, ".
             "DVDs.DVDName ".
             "FROM Transactions, Orders, Customers, DVDs ".
             "WHERE Orders.OrderID = Transactions.OrderID ".
             "AND Customers.CustID = Orders.CustID ".
             "AND DVDs.DVDID = Transactions.DVDID ".
             "ORDER BY  Transactions.OrderID DESC, Customers.CustLN ASC, ".
             "Customers.CustFN ASC, ".
             "Transactions.DateDue DESC, DVDs.DVDName ASC;";

// Execute the SQL query
$result = mysql_query($selectSql, $link);

if(!$result)
   die("Invalid query: ".mysql_error()."<br>".$selectSql);
```

4. The next step in your application is to loop through the results returned by your query. Add the following code to your application file:

```
// Loop through the result set
while($row = mysql_fetch_array($result))
{
// Retrieve the columns from the result set into local variables
   $transId = $row["TransID"];
   $orderId = $row["OrderID"];
   $dvdId = $row["DVDID"];
   $dateOut = $row["DateOut"];
   $dateDue = $row["DateDue"];
   $dateIn = $row["DateIn"];
   $custFirstName = $row["CustFN"];
   $custLastName = $row["CustLN"];
   $dvdName = $row["DVDName"];
```

5. Now put the customer names and dates in a more readable format. Add the following code to your page:

```
   // Convert nulls to empty strings and format the data
   $customerName = "";
   $dateOutPrint = "";
   $dateDuePrint = "";
   $dateInPrint = "";

   if($custFirstName != null)
      $customerName .= $custFirstName." ";

   if($custLastName != null)
      $customerName .= $custLastName;

   if($dvdName == null)
      $dvdName = "";

   $dateFormat = "m-d-Y";

   if($dateOut != null)
      $dateOutPrint = date($dateFormat, strtotime($dateOut));

   if($dateDue != null)
      $dateDuePrint = date($dateFormat, strtotime($dateDue));

   if(strcmp($dateIn, "0000-00-00") != 0)
      $dateInPrint = date($dateFormat, strtotime($dateIn));
```

6. Next, insert the values retrieved from the database in an HTML table structure. Add the following code to the PHP file:

```
// Print each value in each row in the HTML table
?>
      <tr height="35" valign="top">
         <td class="item">
            <nobr>
            <?printf($orderId);?>
```

```
            </nobr>
         </td>
         <td class="item">
            <nobr>
            <?printf($customerName);?>
            </nobr>
         </td>
         <td class="item">
            <nobr>
            <?printf("$dvdName");?>
            </nobr>
         </td>
         <td class="item">
            <nobr>
            <?printf($dateOutPrint);?>
            </nobr>
         </td>
         <td class="item">
            <nobr>
            <?printf($dateDuePrint);?>
            </nobr>
         </td>
         <td class="item">
            <nobr>
            <?printf($dateInPrint);?>
            </nobr>
         </td>
      </tr>
<?
}
```

7. The final section of the file closes the connection and the PHP script. It also closes the `<table>`, `<body>`, and `<html>` elements on the Web page. Add the following code to the end of the PHP file:

```
// Close the database connection and the HTML elements

mysql_close($link);
?>
      </table>
   </td>
</tr>
</table>
</body>
</html>
```

8. Save the index.php file to the appropriate Web application directory.
9. Open your browser, and go to the address `http://localhost/index.php`. (If you saved the file to a different location, use that URL.) Your browser should display a page similar to the one shown in Figure 17-1.

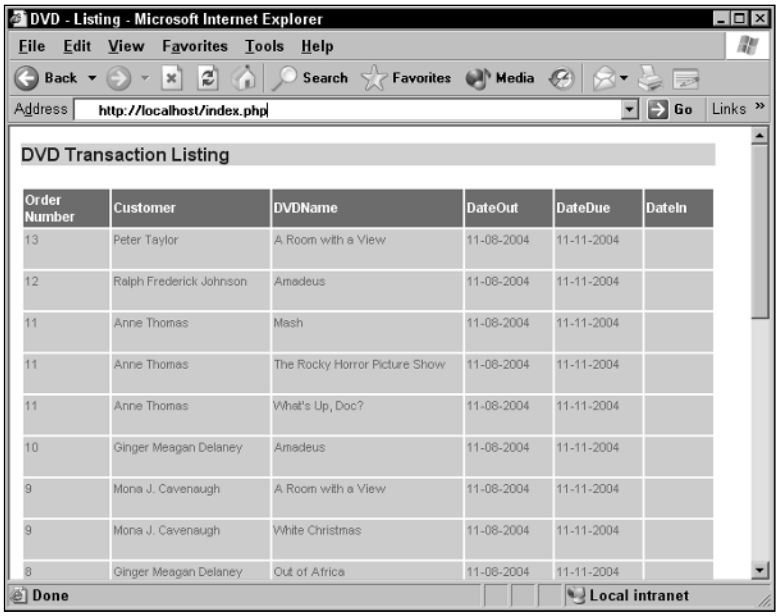

Figure 17-1

If you find that you cannot connect to the MySQL server when trying to open the PHP file, it might be because of the password encryption method used for the MySQL user account. Beginning with MySQL 4.1, a different method is used to encrypt passwords than was used in previous versions. Older versions of PHP (and other client applications) have not yet implemented this new encryption method. As a result, when you try to pass the password from the PHP application to MySQL, there is an encryption mismatch. You can test whether this is a problem by using the mysql client utility — logging in with the mysqlapp user account name and the pw1 password — to access the MySQL server. If you're able to log on to the server with mysql utility, then you know that the account is working fine; in that case, encryption mismatch is probably the problem. To remedy this, open the mysql client utility as the root user, and execute the following SQL statement:

```
SET PASSWORD FOR 'mysqlapp'@'localhost' = OLD_PASSWORD('pw1');
```

The `OLD_PASSWORD()` function saves that password using the old encryption method, which makes the password compatible with the version of PHP that you're using.

How It Works

In this exercise, the first step that you took was to set up the opening HTML section of your index.php file. The `<head>` section establishes the necessary links to the dvdstyle.css and dvdrentals.js files. You then added a `<body>` section that includes two HTML `<table>` elements. The first table provides a structure for the page title — DVD Transaction Listing — and the second table provides the structure for the data to be displayed on the page. The data includes the Order number, customer name, DVD name, and dates that the DVD was checked out, when it is due back, and, if applicable, when it was returned. As a result, the initial table structure created in this section sets up a row for the table headings and a cell for each heading.

For more information about HTML, file linking from HTML, style sheets, and JavaScript functions, refer to the appropriate documentation.

Once you set up your HTML structure on your Web page, you began the PHP section of the page, you established a connection to the MySQL server, and you selected a database, as shown in the following PHP script:

```
<?
$link = mysql_connect("localhost", "mysqlapp", "pw1")
   or die("Could not connect: " . mysql_error());
mysql_select_db("DVDRentals", $link)
   or die("Unable to select database: . mysql_error()");
```

First, you opened the PHP section by using the PHP opening tag (`<?`), which tells the PHP preprocessor that anything enclosed in the opening tag and the closing tag (`?>`) should be processed as PHP script. Next, you defined the `$link` variable based on the `mysql_connect()` function, which establishes a connection to the MySQL server on the local host, using the mysqlapp user account. In the statement that defines the connection, you included an or operator in case a connection cannot be established, in which case the `die()` function is executed. The `die()` function returns the "Could not connect:" message along with the actual MySQL error message. To retrieve the MySQL error message, you used the `mysql_error()` function.

Once you created the `$link` variable, you then used the `mysql_select_db()` function to select the DVDRentals database and again used the or operator to specify the `die()` function, in case the database cannot be selected.

After you established your connections, you then initialized the `$selectSql` variable to equal a `SELECT` statement, as shown in the following statement:

```
$selectSql = "SELECT ".
             "Transactions.TransID, ".
             "Transactions.OrderID, ".
             "Transactions.DVDID, ".
             "Transactions.DateOut, ".
             "Transactions.DateDue, ".
             "Transactions.DateIn, ".
             "Customers.CustFN, ".
             "Customers.CustLN, ".
             "DVDs.DVDName ".
             "FROM Transactions, Orders, Customers, DVDs ".
             "WHERE Orders.OrderID = Transactions.OrderID ".
             "AND Customers.CustID = Orders.CustID ".
             "AND DVDs.DVDID = Transactions.DVDID ".
             "ORDER BY  Transactions.OrderID DESC, Customers.CustLN ASC, ".
             "Customers.CustFN ASC, ".
             "Transactions.DateDue DESC, DVDs.DVDName ASC;";
```

As you can see, this is a basic `SELECT` statement that joins the Transactions, Orders, Customers, and DVDs tables in the DVDRentals database. You then used the `$selectSql` variable as a parameter in the `mysql_query()` function to submit the `SELECT` statement to the MySQL server:

```
$result = mysql_query($selectSql, $link);

if(!$result)
   die("Invalid query: ".mysql_error()."<br>".$selectSql);
```

You initialized the `$result` variable with the query results returned by the `mysql_query()` function (and subsequently the `SELECT` statement). As a precaution, you used the `$result` variable in an `if` statement to verify that a result set was actually returned. The `if` statement condition precedes the `$result` variable with an exclamation point (!), indicating that the condition evaluates to true if the variable contains no result set. As a result, if the result set is empty, the `die()` function is executed, and an error message is returned, along with the original `SELECT` statement (which is returned by the `$selectSql` variable).

Once you check for errors, you then use a `while` loop and the `mysql_fetch_array()` function to process the results returned by your query:

```
while($row = mysql_fetch_array($result))
{

   $transId = $row["TransID"];
   $orderId = $row["OrderID"];
   $dvdId = $row["DVDID"];
   $dateOut = $row["DateOut"];
   $dateDue = $row["DateDue"];
   $dateIn = $row["DateIn"];
   $custFirstName = $row["CustFN"];
   $custLastName = $row["CustLN"];
   $dvdName = $row["DVDName"];
```

The `mysql_fetch_array()` function allows you to process the result set by individual rows. The `while` loop executes the statements in the `while` construction for each row returned by the `mysql_fetch_array()` function. Each row returned by the function is placed in the `$row` variable, which is then used to assign values to individual variables. Each time a `while` loop executes, a variable is assigned a column value from the current row. The `while` loop continues to execute until no more rows are returned by the `mysql_fetch_array()` function.

After you assigned values to the variables, you formatted some of the variable values to make them easier to read. To support this process, you first initialized several new variables to prepare them to hold the formatted content. For example, you initialized the following `$customerName` variable as an empty string (indicated by the pair of double quotes):

```
$customerName = "";
```

The reason you did this is to ensure that if any part of a customer name is `NULL`, that `NULL` value is not displayed. That way, when you begin formatting the names, you can test for the existence of `NULL` values and assign names to the `$customerName` variable only if the values are not `NULL`. If they are `NULL`, then only a string is displayed or only one name, with no `NULL` values appended to the name. As a result, after you initiated the variables, you then began concatenating the names, starting with the first name, as shown in the following if statement:

```
if($custFirstName != null)
   $customerName .= $custFirstName." ";
```

The statement first checks whether the customer's first name is not NULL. If the condition evaluates to true, the value in the $custFirstName variable, plus a space (enclosed in the pair of double quotes), is added to the $customerName variable. Note that when a period precedes an equal sign, the existing variable value is added to the new values, rather than being replaced by those values. This is better illustrated by the next if statement, which then adds the last name to the first name:

```
if($custLastName != null)
   $customerName .= $custLastName;
```

In this case, unless the first name was NULL, the $customerName variable holds the first name value, along with a space, and that is added to the last name value. As a result, the $customerName variable now holds the customer's full name, displayed as first name, space, then last name.

You also used an if statement to ensure that the $dvdName variable contains a string, rather than a NULL value, if a NULL had been returned by the query, as shown in the following statement:

```
if($dvdName == null)
   $dvdName = "";
```

The reason for this is again to ensure that a NULL value is not displayed on the Web page, but rather a blank value if the DVD name is blank.

Next, you formatted the dates so that they're displayed in a more readable format. To support this format, you first initialized a variable with a defined format, and then you used that format to retrieve the data in the specified format. For example, the following statements take the $dateOut value and convert it in a more readable format:

```
$dateFormat = "m-d-Y";

if($dateOut != null)
   $dateOutPrint = date($dateFormat, strtotime($dateOut));
```

To display the date in the format that you want, you must use two PHP functions. The strtotime() functions converts the value stored in the $dateOut variable to a numerical format that can then be used by the date() function. The date() function then extracts the date from the strtotime() output and converts it to the format specified in the $dateFormat function. The output from the date() function is then held in the $dateOutPrint variable.

For one of the date values, you also used the strcmp() function as a condition in an if statement, as shown in the following code:

```
if(strcmp($dateIn, "0000-00-00") != 0)
   $dateInPrint = date($dateFormat, strtotime($dateIn));
```

Recalling the database design of the Transactions table, the DateIn column holds the date that the DVD is returned. That column, though, does not permit NULL values, so the default value is 0000-00-00, which is not a value you would want to display on your Web page. This is one reason why the $dateInPrint is initialized as an empty string. If 0000-00-00 is returned, it is converted to an empty string, and a blank value is displayed on the Web page. Because of the preceding if statement, the value is converted, using the date() and strtotime() functions, if anything other than 0000-00-00 is returned.

Once you have formatted the values in the way that you want, you can then display those values in an HTML table structure. This structure follows the same structure that is defined at the beginning of the file, thus providing the column heads for the rows that you now add. Keep in mind that you are still working in the `while` loop created earlier. Every step that you take at this point still applies to each row returned by the `mysql_fetch_array()` function.

To create the necessary row in the table, you used the PHP closing tag (`?>`) to get out of PHP mode and back into HTML mode. You then created a cell definition for each value that is returned by the result set (and subsequently held in variables). For example, you used the following definition for the first cell in the first row of the table (not counting the heading):

```
<tr height="35" valign="top">
   <td class="item">
      <nobr>
      <?printf($orderId);?>
      </nobr>
   </td>
```

You first used a `<tr>` element to start the row and then `<td>` and `</td>` elements to enclose the individual cell in the row. The `<nobr>` and `</nobr>` elements indicate that there should be no line break between the two tags. Notice that squeezed between all that is PHP script that is enclosed by opening and closing PHP tags. The script is the `printf()` function, which indicates that the value of the `$orderId` variable should be printed to the screen. As a result, the `$orderId` value is displayed in that cell.

This process is repeated for each value in the row, and it is repeated for each row until the `while` statement loops through all the rows in the result set. After the `while` loop, you closed your connection to the MySQL server:

```
mysql_close($link);
```

To close the connection, you used the `mysql_close()` function and specified the `$link` variable (which you initialized at the beginning of the file) to specify which connection to close.

As the insert.php file currently exists, your application does nothing but display a list of transactions in the DVDRentals database. In most cases, you want your applications to do more than that. For this reason, you now learn how to use PHP to insert data in your database.

Inserting and Updating Data in a MySQL Database

Many of the concepts that you learned when retrieving data from your database can be also be applied to inserting data. The main difference is that you do not have to process a result set; however, you must still set up your SQL statement and use the `mysql_query()` function to execute the statement.

For instance, suppose that you want your application to insert data in a table named CDs. You can start by defining a variable to hold the `INSERT` statement, as shown in the following example:

```
$insertSql = "INSERT INTO CDs (CDName, InStock, OnOrder, Category) VALUES
('Beethoven Symphony No. 6 Pastoral', 10, 10, 'classical')";
```

As you can see, this is a typical `INSERT` statement. Once it is assigned to the variable, you can use the `mysql_query()` function to execute that statement:

```
mysql_query($selectSql, $link);
   or die("Invalid query: " . mysql_error() . "<br>".$selectSql);
```

If the statement executes successfully, the data is inserted in the appropriate table. If the statement fails, the `die()` function executes, and an error message is returned.

Updating data in the database is very similar to inserting data. You must define your `UPDATE` statement and then execute it. For example, you can assign an `UPDATE` statement to the `$updateSql` variable, as shown in the following code:

```
$updateSql = "UPDATE CDs SET InStock=5 WHERE CDID = 10";
```

Once again, you can use your variable in the `mysql_query()` function to send the statement to the MySQL server:

```
mysql_query($updateSql, $link);
   or die("Invalid query: " . mysql_error() . "<br>".$updateSql);
```

As the statement shows, the data is either updated or an error is returned. You can use these data insert and update methods — along with a number of other methods that you've seen — to create an application far more robust than simply displaying data. Return now to your DVDRentals application.

Adding Insert and Update Functionality to Your Application

So far, your DVDRentals application displays only transaction-related information. In this section, you add to the application so that it also allows you to add a transaction to an existing order and to modify transactions. To support the added functionality, you must create three more files — edit.php, insert.php, and update.php — and you must modify the index.php file. Keep in mind that your index.php file acts as a foundation for the rest of the application. As a result, you should be able to add a transaction and edit a transaction by first opening the index.php file and then maneuvering to wherever you need to be in order to accomplish these tasks.

The first additional file that you create is the edit.php file. The file serves two roles: adding transactions to existing orders and editing existing transactions. These two operations share much of the same functionality, so combining them in one file saves duplicating code. If you're adding a new transaction, the Web page will include a drop-down list that displays each order number and the customer associated with that order, a drop-down list that displays DVD titles, a text box for the date the DVD is rented, and a text box for the date the DVD should be returned. The default value for the date rented text box is the current date. The default value for the date due text box is three days from the current date.

If you're editing a transaction, the Web page will display the current order number and customer, the rented DVD, the date the DVD was rented, the date it's due back, and, if applicable, the date that the DVD was returned.

The edit.php Web page will also contain two buttons: Save and Cancel. The Save button saves the new or updated record and returns the user to the index.php page. The Close button cancels the operation, without making any changes, and returns the user to the index.php page.

After you create the edit.php page, you then create the insert.php file and the update.php file in the Try It Out sections that follow this one. From there, you modify the index.php page to link together the different functionalities. Now take a look at how to create the edit.php file.

Try It Out Creating the edit.php File

The following steps describe how to create the edit.php file, which supports adding new transactions to a record and editing existing transactions:

1. As you did with the index.php file, you must establish a connection to the MySQL server and select the database. Use a text editor to start a new file, and enter the following code:

```
<?
// Connect to server or return an error
$link = mysql_connect("localhost", "mysqlapp", "pw1")
or die("Could not connect: " . mysql_error());

// Select a database or return an error
mysql_select_db("DVDRentals", $link);
```

   For security purposes and to streamline your code, you might want to store your connection information in a different file and then call that file from the PHP page. If you take an approach similar to this, you must still supply the necessary connection parameters.

2. To support error-checking operations, you need to initialize a variable and set its value to an empty string. Add the following statement your file:

```
// Initialize an error-related variable
$error = "";
```

3. Next, you want to put a mechanism in place that assigns posted values to the appropriate variables. Add the following code to the edit.php file:

```
// Initialize variables with parameters retrieved from the posted form
if(isset($_POST["command"]))
   $command = $_POST["command"];
if(isset($_POST["transaction_id"]))
   $transactionId = $_POST["transaction_id"];
if(isset($_POST["date_due"]))
   $dateDue = $_POST["date_due"];
if(isset($_POST["order_id"]))
   $orderId = $_POST["order_id"];
if(isset($_POST["dvd_id"]))
   $dvdId = $_POST["dvd_id"];
if(isset($_POST["date_out"]))
   $dateOut = $_POST["date_out"];
if(isset($_POST["date_in"]))
   $dateIn = $_POST["date_in"];
```

4. Next, the file should process the new or edited transaction when the user clicks the Save button. First, you must check to see whether the form is complete before you try to process the values. Enter the following error-related code:

```
// Process the save and savenew commands
if((strcmp("save", $command) == 0) || (strcmp("savenew", $command) == 0))
{
// Check for missing parameters
   if($orderId ==  -1)
      $error .= "Please select an \"Order\"<br>";

   if($dvdId ==  -1)
      $error .= "Please select a \"DVD\"<br>";

   if(($dateOut == null) || (strlen($dateOut) == 0))
      $error .= "Please enter a \"Date Out\" Value<br>";

   if(($dateDue == null) || (strlen($dateDue) == 0))
      $error .= "Please enter a \"Date Due\" Value<br>";
```

Note that the application does not check the format of the date submitted by users. Normally, an application would include some type of mechanism to ensure that submitted dates are in a usable format.

5. Next, you should format the date values to ensure that they can be inserted in the database. Then you can carry out the update or insert by calling the applicable include files. (These files are created in later Try It Out sections.) Enter the following code in your file:

```
   if(strlen($error) == 0)
   {

// Reformat dates so that they are compatible with the MySQL format
      if($dateOut != null)
          $dateOut = substr($dateOut, 6, 4)."-".substr($dateOut, 0, 2)."-
".substr($dateOut, 3, 2);

      if($dateDue != null)
          $dateDue = substr($dateDue, 6, 4)."-".substr($dateDue, 0, 2)."-
".substr($dateDue, 3, 2);

      if($dateIn != null)
          $dateIn = substr($dateIn, 6, 4)."-".substr($dateIn, 0, 2)."-
".substr($dateIn, 3, 2);
      else
          $dateIn ="0000-00-00";

      if(strcmp("save", $command) == 0)
      {
// Run the update in update.php
         include "update.php";
      }
      else
      {
// Run the insert in insert.php
```

```
         include "insert.php";
      }

// Redirect the application to the listing page
      header("Location: index.php");
      exit;
   }
}
```

6. The next step is to set up the file to support adding or updating a record when the user has been redirected to this page from the index.php page. This is done as part of the `else` statement in an `if...else` structure. This particular section sets up the default values for a new record. Add the following code to your file:

```
else
{
// If it is a new record, initialize the variables to default values
   if(strcmp("add", $command) == 0)
   {
      $transactionId = 0;
      $orderId = 0;
      $dvdId = 0;
      $dateOut = date("m-d-Y", time());
      $dateDue = date("m-d-Y", time() + 3*24*60*60);
      $dateIn = "";
   }
```

7. Next, you must set up the file with the values necessary to support editing a record. This involves retrieving records to set an initial value for a number of variables. Add the following code to your PHP file:

```
   else
   {
// If it is an existing record, read from database
      if($transactionId != null)
      {
// Build query from transactionId value passed down from form
         $selectSql = "SELECT ".
                      "OrderID, ".
                      "DVDID, ".
                      "DateOut, ".
                      "DateDue, ".
                      "DateIn ".
                      "FROM Transactions ".
                      "WHERE TransID = '$transactionId'";

// Execute query
         $result = mysql_query($selectSql, $link);

         if (!$result)
            die("Invalid query: " . mysql_error(). "<br>".$selectSql);

// Populate the variables for display into the form
```

```
            if($row = mysql_fetch_array($result))
            {
               $orderId = $row["OrderID"];
               $dvdId = $row["DVDID"];
               $dateOut = $row["DateOut"];
               $dateDue = $row["DateDue"];
               $dateIn = $row["DateIn"];

   // Reformat the dates into a more readable form
               if($dateOut != null)
                  $dateOut = substr($dateOut, 5, 2)."-".substr($dateOut, 8, 2)."-
   ".substr($dateOut, 0, 4);
               else
                  $dateOut = "";

               if($dateDue != null)
                  $dateDue = substr($dateDue, 5, 2)."-".substr($dateDue, 8, 2)."-
   ".substr($dateDue, 0, 4);
               else
                  $dateDue = "";

               if($dateIn != "0000-00-00")
                  $dateIn = substr($dateIn, 5, 2)."-".substr($dateIn, 8, 2)."-
   ".substr($dateIn, 0, 4);
               else
                  $dateIn = "";
            }
         }
      }
   }
   ?>
```

8. Now you must create the HTML section of your form to allow users to view and enter data. This section includes a form to pass data to PHP and the table structure to display the form. Add the following code to your PHP file:

```
<html>
<head>
   <title>DVD - Listing</title>
   <link rel="stylesheet" href="dvdstyle.css" type="text/css">
   <script language="JavaScript" src="dvdrentals.js"></script>
</head>

<body>
<form name="mainForm" method="post" action="edit.php">
<input type="hidden" name="command" value="view">
<input type="hidden" name="transaction_id" value="<?printf($transactionId);?>"
size="50">

<p></p>

<table cellspacing="0" cellPadding="0" width="619" border="0">
<tr>
```

```
<td>
<table height="20" cellspacing="0" cellPadding="0" width="619" bgcolor="#bed8e1"
border="0">
<tr align=left>
   <td valign="bottom" width="400" class="title">
      DVD Transaction
   </td>
   <td align="right" width="219" class="title"> </td>
</tr>
</table>
<br>
<?if(strlen($error) > 0){?>
<table cellspacing="2" cellPadding="2" width="619" border="0">
<tr>
   <td width="619" class="error">
      <?printf($error);?>
   </td>
</tr>
</table>
<?}?>
```

9. Now create the first row of your form, which allows users to view and select an order ID. Enter the following code in your file:

```
<table cellspacing="2" cellPadding="2" width="619" border="0">
<tr>
   <td width="250" class="heading">Order</td>
   <td class="item">
      <select name="order_id">
         <option value="-1">Select Order</option>
<?
// Retrieve data to populate drop-down list
   $selectSql = "SELECT ".
                "Orders.OrderID, ".
                "Orders.CustID, ".
                "Customers.CustFN, ".
                "Customers.CustLN ".
                "FROM Orders, Customers ".
                "WHERE ".
                "Customers.CustID = Orders.CustID ".
                "ORDER BY Orders.OrderID DESC, ".
                "Customers.CustLN ASC, Customers.CustFN ASC";

// Execute the query
$result = mysql_query($selectSql, $link);

if (!$result)
   die("Invalid query: " . mysql_error(). "<br>".$selectSql);

// Loop through the results
while($row = mysql_fetch_array($result))
{
// Assign returned values to the variables
```

```
      $orderId1 = $row["OrderID"];
      $custFirstName = $row["CustFN"];
      $custLastName = $row["CustLN"];

   // Format the data for display
      $customerName = "";

      if($custFirstName != null)
         $customerName .= $custFirstName . " ";

      if($custLastName != null)
         $customerName .= $custLastName;

   // If the order id matches the existing value, mark it as selected
      if($orderId1 != $orderId)
      {
   ?>
         <option value="<?printf($orderId1)?>"><?printf($orderId1);?> -
   <?printf($customerName);?></option>
   <?
      }
      else
      {
   ?>
         <option selected value="<?printf($orderId1);?>"><?printf($orderId1)?> -
   <?printf($customerName);?></option>
   <?
      }
   }
   ?>
         </select>
      </td>
   </tr>
```

10. The second row of your form allows users to view and select a DVD to associate with your transaction. Add the following code to your edit.php file:

```
   <tr>
      <td class="heading">DVD</td>
      <td class="item">
         <select name="dvd_id">
            <option value="-1">Select DVD</option>
   <?
   // Retrieve data to populate drop-down list
   $selectSql = "SELECT DVDID, DVDName FROM DVDs ORDER BY DVDName";

   $result = mysql_query($selectSql, $link);

   if (!$result)
      die("Invalid query: " . mysql_error(). "<br>".$selectSql);

   while($row = mysql_fetch_array($result))
   {
```

```
   $dvdId1 = $row["DVDID"];
    $dvdName = $row["DVDName"];

    if($dvdName == null)
       $dvdName = "";

    if($dvdId1 != $dvdId)
    {
?>
          <option value="<?printf($dvdId1);?>"><?printf($dvdName);?></option>
<?
    }
    else
    {
?>
          <option selected
value="<?printf($dvdId1);?>"><?printf($dvdName);?></option>
<?
    }
}
?>
      </select>
    </td>
</tr>
```

11. Next, create three more rows in your table, one for each date-related value. Enter the following code:

```
<tr>
    <td class="heading">Date Out</td>
    <td class="item">
       <input type="text" name="date_out" value="<?printf($dateOut);?>" size="50">
    </td>
</tr>
<tr>
    <td class="heading">Date Due</td>
    <td class="item">
       <input type="text" name="date_due" value="<?printf($dateDue);?>" size="50">
    </td>
</tr>
<?if((strcmp("add", $command) != 0) && (strcmp("savenew", $command) != 0)){?>
<tr>
    <td class="heading">Date In</td>
    <td class="item">
       <input type="text" name="date_in"  value="<?printf($dateIn);?>" size="50">
    </td>
</tr>
<?}?>
```

12. Now add the Save and Cancel buttons to your form by appending the following code to your file:

```
<tr>
    <td colspan="2" class="item" align="center">
       <table cellspacing="2" cellPadding="2" width="619" border="0">
```

```
      <tr>
         <td align="center">
            <?if((strcmp("add", $command) == 0) || (strcmp("savenew", $command) ==
0)){?>
            <input type="button" value="Save" class="add" onclick="doSave(this,
'savenew')">
            <?}else{?>
            <input type="button" value="Save" class="add" onclick="doSave(this,
'save')">
            <?}?>
         </td>
         <td align="center">
            <input type="button" value="Cancel" class="add"
onclick="doCancel(this)">
         </td>
      </tr>
      </table>
   </td>
</tr>
```

13. Close the various HTML elements, and close your connection to the MySQL server by entering the following code:

```
</table>
</form>
</body>
</html>

<?
// Close connection
mysql_close($link);
?>
```

14. Save the edit.php file to the appropriate Web application directory.

How It Works

In this exercise, you created the edit.php file, which supports the insert and update functionality in your DVDRentals application. The first code that you added to the file established the connection to the MySQL server and selected the DVDRentals database, just as it did in the index.php file. After you established your connection, you initiated the `$error` variable by setting its value to an empty string, as shown in the following statement:

```
$error = "";
```

The variable is used to display error messages if a user does not fill out the form properly. You set the value to an empty string so that you can use that to verify whether any error messages have been received, as you see later in the page. After you initiated the `$error` variable, you initiated a number of other variables based on values returned to the `$_POST` array. For example, the following `if` statement initiates the `$command` variable:

```
if(isset($_POST["command"]))
   $command = $_POST["command"];
```

The `if` statement first uses the `isset()` function to determine whether the `$_POST` array contains a command value. If it does, then that value is assigned to the variable. The `$_POST` array contains those values that are posted to the page when a form is submitted. For the command and transaction_id values in the `$_POST` array, the initial values are derived from a form on the index.php page. As you see later in the application development process, when you access the edit.php page from the index.php page (through the click of a button), the index.php form submits these values to the edit.php page.

After you assigned `$_POST` array values to your variables, you set up `if...else` statements that begin with the following if condition:

```
if((strcmp("save", $command) == 0) || (strcmp("savenew", $command) == 0))
```

The `if` statement specifies two conditions. The first condition uses the `strcmp()` function to compare the string save to the current value in the `$command` variable. If the values are equal (`== 0`), the condition evaluates to true. The second condition also uses the `strcmp()` function to compare the string `savenew` to `$command`. If the values are equal, the condition evaluates to true. Because the conditions are connected by the `or` (`||`) operator, either condition can be true in order for the if statement to be executed. If neither condition is true, the `else` statement is executed.

> *The* `save` *and* `savenew` *command values are issued when you click the Save button. You learn more about that button shortly.*

The `if...else` statements contain a number of `if...else` statements embedded in them. Before getting deeper into the code that makes up all these statements, first take a look at a high overview of the logic behind these statements. It gives you a better sense of the bigger picture and should make understanding the individual components a little easier. The following pseudo-code provides an abbreviated statement structure starting with the outer `if` statement described previously:

```
if $command = save or savenew, continue (if !=, go to else)
{
   if incorrect form entry, return error and redisplay page (x 4)
   if no error, continue
   {
      if date != null, format date (x 2)
      if date != null, format date; else format date to 0000-00-00
      if $command = save, include update.php; else include insert.php
      redirect to index.php
      exit edit.php
   }
}
else (if $command != save or savenew)
{
   if $command = add, continue (if !=, go to else)
   {
      initialize variables (x 6)
   }
   else (if $command != add)
   {
      if $transactionId != null
      {
         process query
         if query results exist, fetch results
```

```
            {
               assign variables (x 5)
               if date != null, format date; else set date to empty string (x 2)
               if date != 0000-00-00, format date; else set date to empty string
            }
         }
      }
   }
```

As you can see, the action taken depends on the values in the `$command` and `$transactionId` variables. The `if` statement basically determines what happens when you try to save a record, and the `else` statement determines what happens when you first link to the page from the index.php page. Embedded in the `else` statement is another set of `if...else` statements. The embedded `if` statement determines what happens if you want to add a transaction, and the embedded `else` statement determines what happens if you want to update a transaction.

To better understand all this, you now take a closer look at some of the statements in this `if...else` construction. You've already seen the opening `if` statement. Following this statement are four embedded `if` statements that handle errors if a user does not properly fill out the form on the page. For example, the first of these statements sets up a condition to return an error if it doesn't have a proper `$orderId` value:

```
if($orderId ==  -1)
   $error .= "Please select an \"Order\"<br>";
```

The `if` statement specifies that the condition is true of the `$orderId` value equals -1. If this occurs, the `$error` variable is appended. Notice that a period/equal sign (`.=`) follows the variable name, meaning that any message received is appended to whatever the current variable value is (rather than replacing that value). This allows multiple error messages to be returned, if more than one error occurs.

The four error-related `if` statements all work in a similar manner as the one shown previously. After you defined these statements, you added the following `if` statement, which is also related to errors:

```
if (strlen($error) == 0)
```

The `if` statement specifies that the current value for the `$error` variable must contain no characters in order for the condition to evaluate to true. In other words, if there is no error message, continue; otherwise, stop processing this part of the PHP script (which includes everything to the end of the `if...else` statements). Assuming that there are no errors, PHP continues processing the `if` statement by formatting the date values so that they are in a form compatible with MySQL. For example, the following if statement formats the value in the `$dateOut` variable:

```
if($dateOut != null)
   $dateOut = substr($dateOut, 6, 4)."-".substr($dateOut, 0, 2)."-
".substr($dateOut, 3, 2);
```

If the condition evaluates to true, the `substr()` function is used to extract specific characters from the `$dateOut` value. For example, the first `substr()` function extracts four characters starting with the sixth character. If the date in the application appears as 10-09-2004, the result will be 2004. This is then followed by a dash and two additional `substr()` functions. In the end, 10-09-2004 is converted to 2004-10-09 so that it can be inserted in the database.

Once the dates have been properly formatted, you then referenced the necessary include files that contain the update and insert code. (You create these files in later Try It Out sections.) You referenced the include files in the following `if...else` statements:

```
if(strcmp("save", $command) == 0)
{
   include "update.php";
}
else
{
   include "insert.php";
}
```

The `if` statement specifies that the `$command` variable must equal `save`. If this is the case, then the update.php file is called. Otherwise, the insert.php file is called, as indicated in the else statement. Once the statements in the include file are executed (and the database is updated), the `header()` function is executed, followed by the exit command:

```
header("Location: index.php");
exit;
```

The `header()` function redirects the user to the index.php page, and the `exit` command exits the current page. This completes the outer `if` statement that initiates this section of code. If the `if` statement is not applicable (`$command` does not equal `save` or `savenew`), PHP executes the `else` statement.

The `else` statement is made up of its own embedded `if...else` statements. The `if` statement applies if the `$command` variable currently holds the value `add`, as shown in the following code:

```
if(strcmp("add", $command) == 0)
{
   $transactionId = 0;
   $orderId = 0;
   $dvdId = 0;
   $dateOut = date("m-d-Y", time());
   $dateDue = date("m-d-Y", time() + 3*24*60*60);
   $dateIn = "";
```

If `$command` is set to `add`, this means that you are creating a new transaction. To prepare the Web page with the variables it needs to display the form properly when you open the page, you set a number of variables to specific values. For example, you set `$transactionId` to 0, which is an unused number so it does not match any current transaction IDs.

You also used the `date()` function to set default date values. For the `$dateOut` variable, you set a date that is based on `time()` function, which returns the current timestamp. The `date()` function extracts just the date part of the timestamp and puts it into the format defined by the first parameter (`m-d-Y`). The `$dateDue` variable is similar to the `$dateOut` variable except that it adds three days to the current timestamp (3 * 24 hours * 60 minutes * 60 seconds). The `$dateIn` variable is set to an empty string so that only a blank is displayed.

In the embedded `else` statement, you included an `if` statement that verifies that the `$transactionId` variable is not null, as shown in the following:

```
if($transactionId != null)
```

If the value is not null (a value does exist), the remaining part of the `if` statement is executed. This means that you are editing an existing transaction, in which case, the `$transactionId` variable identifies that transaction. You then added the code necessary to retrieve data from the database (based on the `$transactionId` value), which you then used to define the variables used to display the form. Because the user is editing an existing transaction at this point, the form should include the current order number, DVD ID, and dates. You then formatted the days so that they are properly displayed.

Once you set up the data to populate the form, you have completed the `if...else` block of code. If necessary, refer back to the summary code that is provided at the beginning of this description. This provides a handy overview of what is going on.

The next section of code that you created set up the HTML for the Web page. As with the index.php file, the HTML section includes header information that links to the dvdstyle.css file and the dvdrentals.js file. The section also includes a `<form>` element and two `<input>` elements:

```
<form name="mainForm" method="post" action="edit.php">
<input type="hidden" name="command" value="view">
<input type="hidden" name="transaction_id" value="<?printf($transactionId);?>"
size="50">
```

A form is an HTML structure that allows a user to enter or select data and then submit that data. That data can then be passed on to other Web pages or to PHP script. This particular form uses the `post` method to send data (`method="post"`) and sends that data to itself (`action="edit.php"`). Beneath the form, you added two `<input>` elements that create the initial values to be inserted in the `command` and `transaction_id` parameters. The `command` parameter is set to `view`, and the `transaction_id` parameter is set to the value contained in the `$transactionId` variable. To use the variable to set the `transaction_id` value, you must enclose the variable in the PHP opening and closing tags and then use the `printf()` function to print that value to HTML so that it can be used by the form. Also note that the input type for both `<input>` elements is hidden, which means that the user does not actually see these two elements. Instead, they serve only as a way to pass the `command` and `transaction_id` values, which is done in the background. The user does not enter these values.

> *Forms are a common method used in HTML to pass data between pages. They are also useful for passing values between HTML and PHP. For more information about forms, consult the applicable HTML documentation.*

After you defined the form, you then set up the table to display the heading DVD Transaction at the top of the page. From there, you added another table whose purpose is to display error messages, as shown in the following code:

```
<?if(strlen($error) > 0){?>
<table cellspacing="2" cellPadding="2" width="619" border="0">
<tr>
   <td width="619" class="error">
      <?printf($error);?>
```

```
   </td>
</tr>
</table>
<?}?>
```

The HTML table structure is preceded by PHP opening and closing tags so that you can use an `if` statement to specify a condition in which the table is displayed. The `if` statement specifies that the `$error` variable must contain a string that is greater in length than 0. This is done by using the `strlen()` function to determine the length of the `$error` value and then comparing the length to 0. If the condition evaluates to true, the table is created and the error is printed. At the end of the table, you again used a pair of opening and closing tags to add the closing bracket of the `if` statement.

Once you have established a way for error messages to be displayed, you set up the table that is used to display the part of the form that the user sees. The first row of the form will contain a drop-down list of order IDs — along with the customer names associated with those orders — that the user is able to select from when adding a transaction. To populate this drop-down list, you retrieved data from the database, processed the data, and formatted the customer name. The methods used for retrieving and processing the data are the same methods that you've already used in the application. Once you retrieved the value, you added another form element to your Web page; only this form element is visible to the user:

```
   if($orderId1 != $orderId)
   {
?>
      <option value="<?printf($orderId1)?>"><?printf($orderId1);?> -
<?printf($customerName);?></option>
<?
   }
   else
   {
?>
      <option selected value="<?printf($orderId1);?>"><?printf($orderId1)?> -
<?printf($customerName);?></option>
<?
   }
```

The form element shown here is an `<option>` element. An `<option>` element allows a user to select from a list of options in order to submit data to the form. There are actually two `<option>` elements here, but only one is used because PHP `if...else` statements encloses the `<option>` elements. The `if` statement condition specifies that `$orderId1` should not equal `$orderId`. If they are not equal, the `if` statement is executed and the first `<option>` element is used; otherwise, the second `<option>` element is used. The second element includes the selected option, which means that the current order ID is the selected option when the options are displayed.

The `$orderId1` variable receives its value from the results returned by the `SELECT` statement used to populate the `<option>` element. The `$orderId` variable receives its value from the `SELECT` statement that is used to assign values to variables once it has been determined that the user is editing an existing transaction. (This occurs when the `if...else` statements earlier in the code are processed.) If the two values are equal, the second `<option>` element is used, which means that the current order ID is displayed when this page is loaded. If the two values are not equal, which is the condition specified in the `if` statement, no order ID is displayed, which you would expect when creating a new record.

The next row that you created for your form table allows users to select from a list of DVD names. The same logic is used to create the drop-down list available to the users. The only difference is that, because only DVD names are displayed, no special formatting or concatenation is required to display the values.

After you created your two rows that display the drop-down lists to the users, you created three date-related rows. Each row provides a text box in which users can enter the appropriate dates. For example, the first of these rows includes the following form element:

```
<input type="text" name="date_out" value="<?printf($dateOut);?>" size="50">
```

As you can see, this is an `<input>` element similar to the ones that you created when you first defined the form. Only the input type on this one is not hidden, but instead is text, which means that a text box will be displayed. The name of the text box is date_out. This is actually the name of the parameter that will hold the value that is submitted by the user. The initial value displayed in the text box depends on the value of the `$dateOut` variable. For new records, this value is the current date, and for existing records, this is the value as it currently exists in the database. (Both these values are determined in the earlier PHP script.)

Once you completed setting up the various form elements, you added two more elements: one for the Save button and one for the Cancel button. For example, your code for the Save button is as follows:

```
<td align="center">
   <?if((strcmp("add", $command) == 0) || (strcmp("savenew", $command) == 0)){?>
   <input type="button" value="Save" class="add" onclick="doSave(this, 'savenew')">
   <?}else{?>
   <input type="button" value="Save" class="add" onclick="doSave(this, 'save')">
   <?}?>
</td>
```

A button is also an `<input>` element on a form, but the type is specified as a button. The value for this element determines the name that appears on the button, which in this case is Save. The `class` option specifies the style that should be used in the button, as defined in the dvdstyle.css file, and the `onclick` option specifies the action to be taken. In this case, the action is to execute the `doSave()` function, which is defined in the dvdrentals.js file.

Notice that there are again two `<input>` elements, but only one is used. If the command value equals `add` or `savenew`, the first `<input>` element is used; otherwise, the second `<input>` element is used. When you click the Save button, the `doSave()` function is called. The function takes one argument, `this`, which is a self-referencing value that indicates that the action is related to the current HTML input button. When the function is executed, it submits the form to the edit.php page and sets the `command` parameter value to `savenew` or `save`, depending on which `<input>` option is used. Based on the `command` value, the PHP script is processed once again, only this time, the first `if` statement (in the large `if... else` construction) evaluates to true and that statement is executed. Assuming that there are no errors, the date values are reformatted for MySQL, the PHP script in the update.php or insert.php include file is executed, and the user is redirected to the index.php page.

As you can see, the edit.php page provides the main logic that is used to insert and update data. As the code in this page indicates, you must also create the include files necessary to support the actual insertion and deletion of data. In the next Try It Out section, you create the insert.php file. The file contains only that script that is necessary to insert a record, based on the value provided by the user in the edit.php form.

Try It Out Creating the insert.php File

The following steps describe how to create the insert.php file:

1. In your text editor, create a new file, and enter the following code:

```
<?
// Build the INSERT statement
$sqlString = "INSERT INTO Transactions (OrderID, DVDID, DateOut, DateDue) VALUES
($orderId, $dvdId, '$dateOut', '$dateDue')";

// Execute the INSERT statement
mysql_query($sqlString)
   or die("Error in query $sqlString ".mysql_error());
?>
```

2. Save the insert.php file to the appropriate Web application directory.

How It Works

As you can see, this is a very simple file. You first initialized the `$sqlString` variable with the `INSERT` statement necessary to add the record:

```
$sqlString = "INSERT INTO Transactions (OrderID, DVDID, DateOut, DateDue) VALUES
($orderId, $dvdId, '$dateOut', '$dateDue')";
```

Notice that the `INSERT` statement includes the variables that have been assigned values through the process of the user filling out and submitting the form. The `$sqlString` variable is then used in the following `mysql_query()` function to execute the `INSERT` statement:

```
mysql_query($sqlString)
   or die("Error in query $sqlString ".mysql_error());
```

If the statement executes successfully, the rest of the applicable script in the edit.php file is executed and the user is returned to the index.php file. If the `INSERT` statement fails, the user receives an error.

In addition to creating the insert.php file, you must also create the update.php file. This file works just like the insert.php file in that it is included in the edit.php file. This is the same as including these statements directly in the edit.php file. In the following Try it Out section, you create the update.php file.

Try It Out Creating the update.php File

The following steps describe how to create the update.php file:

1. Open a new file in your text editor, and enter the following code:

```
<?
// Build the UPDATE statement
$sqlString = "UPDATE Transactions SET OrderID = $orderId, DVDID = $dvdId, DateOut =
'$dateOut', DateDue = '$dateDue', DateIn = '$dateIn' WHERE TransID =
$transactionId";

// Execute the UPDATE statement
```

```
mysql_query($sqlString)
   or die("Error in query $sqlString " . mysql_error());
?>
```

2. Save the update.php file to the appropriate Web application directory.

How It Works

Once again, you initialized the `$sqlString` variable with an SQL statement. In this case, the statement is an `UPDATE` statement, as shown in the following code:

```
$sqlString = "UPDATE Transactions SET OrderID = $orderId, DVDID = $dvdId, DateOut =
'$dateOut', DateDue = '$dateDue', DateIn = '$dateIn' WHERE TransID =
$transactionId";
```

The `UPDATE` statement uses the variables that were set when the user submitted the form. Once the `$sqlString` variable is set with the `UPDATE` statement, you can use it in the `mysql_query()` function:

```
mysql_query($sqlString)
   or die("Error in query $sqlString " . mysql_error());
```

As you can see, this is the same process that you saw when executing the `INSERT` statement in the previous exercise. You set a variable with the SQL statement, and then you execute the statement by using the `mysql_query()` function. Now only one step remains to set up your application to insert and update data. You must modify the index.php file so that it includes the functionality necessary to link the user to the edit.php page. The following Try It Out section explains how to modify the index.php file. It then walks you through the process of inserting a transaction and then modifying that transaction.

Try It Out — Modifying the index.php File

The following steps describe how to modify the index.php file to support the insert and update operations:

1. In your text editor, open the index.php file. Add a form, an `<input>` element, and a cell definition to your HTML code. Add the following code (shown with the gray screen background) to your file:

```
<html>
<head>
   <title>DVD - Listing</title>
   <link rel="stylesheet" href="dvdstyle.css" type="text/css">
   <script language="JavaScript" src="dvdrentals.js"></script>
</head>

<body>

<form name="mainForm" method="post" action="index.php">
<input type="hidden" name="command" value="view">
<input type="hidden" name="transaction_id" value="">

<p></p>

<table cellSpacing="0" cellPadding="0" width="619" border="0">
```

```
<tr>
   <td>
      <table height="20" cellSpacing="0" cellPadding="0" width="619"
bgcolor="#bed8e1" border="0">
      <tr align=left>
         <td valign="bottom" width="400" class="title">
            DVD Transaction Listing
         </td>
         <td align="right" width="219" class="title">
            <input type="button" value="New Transaction" class="add"
onclick="doAdd(this)">
         </td>
      </tr>
      </table>
      <br>
      <table cellSpacing="2" cellPadding="2" width="619" border="0">
      <tr>
         <td width="250" class="heading">Order Number</td>
         <td width="250" class="heading">Customer</td>
         <td width="250" class="heading">DVDName</td>
         <td width="185" class="heading">DateOut</td>
         <td width="185" class="heading">DateDue</td>
         <td width="185" class="heading">DateIn</td>
         <td width="99" class="heading"> </td>
      </tr>
```

When adding code to your file, be sure to add it in the position shown here.

2. Next, add an HTML table cell and an `<input>` element to the area of code that prints out the values returned by the database. Add the following code (shown with the gray screen background) to your file:

```
         <td class="item">
            <nobr>
            <?printf($dateInPrint);?>
            </nobr>
         </td>
         <td class="item" valign="center" align="center">
            <input type="button" value="Edit" class="edit" onclick="doEdit(this,
<?printf($transId)?>)">
         </td>
      </tr>
```

3. Now you must close the form, which you do near the end of the file. To close the form, you must use a `</form>` element. Add the following code (shown with the gray screen background) to the end of the PHP file:

```
      </table>
   </td>
</tr>
</table>
</form>
</body>
</html>
```

4. Save the index.php file.

5. Open your browser, and go to the address `http://localhost/index.php`. Your browser should display a page similar to the one shown in Figure 17-2.

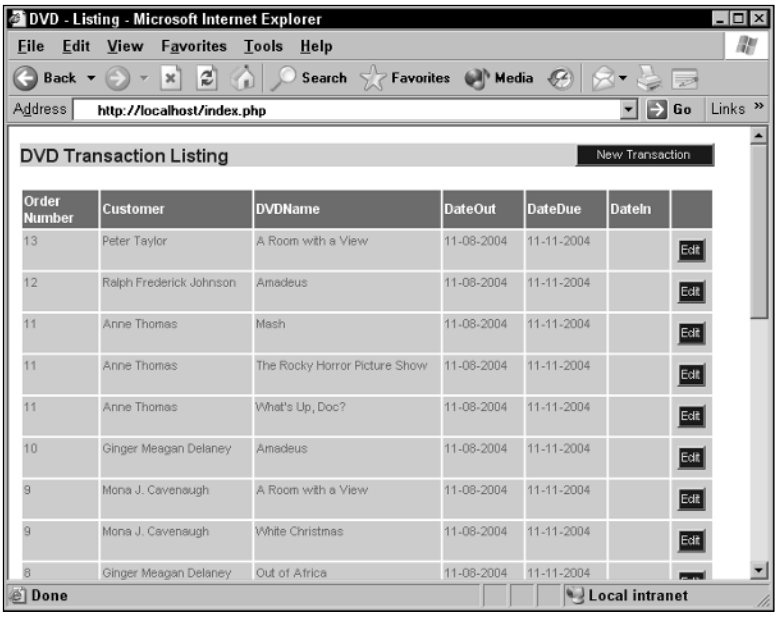

Figure 17-2

6. Click the New Transaction button at the top of the page. Your browser should display a page similar to the one shown in Figure 17-3.

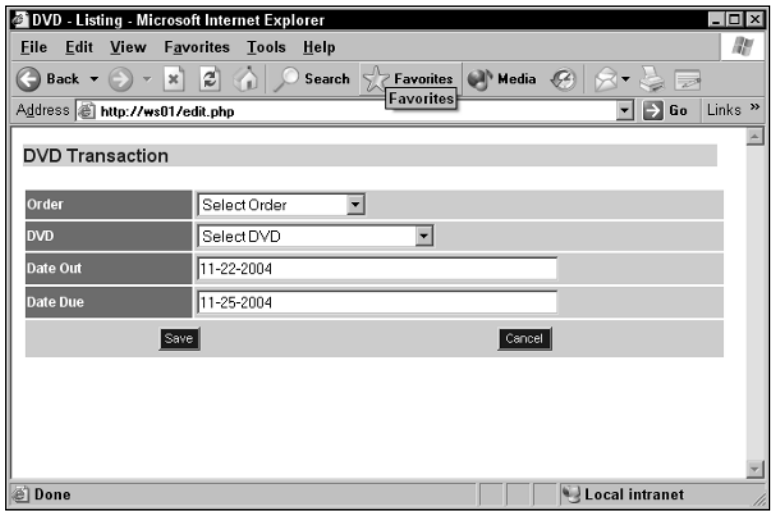

Figure 17-3

7. Now add a transaction to an existing order. In the Order drop-down list, select 13 - Peter Taylor. In the DVD drop-down list, select *Out of Africa*. Click Save. You're returned to the index.php page. The new transaction should now display at the top of the list.

8. Now edit the new transaction. In the row of the transaction that you just added, click the Edit button. Your browser should display a page similar to the one shown in Figure 17-4.

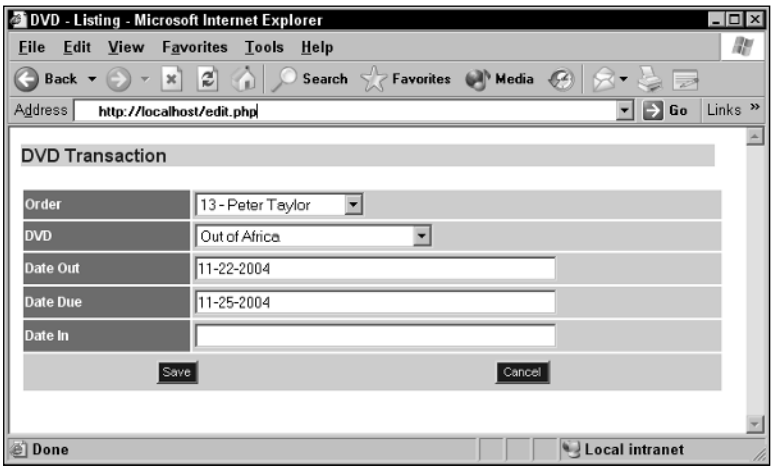

Figure 17-4

9. In the Date In text box, type the same date that is in the Date Due text box. Be certain to type the date in the same format that is used for the other date-related text boxes. Click the Save button. You should be returned to the index.php page.

How It Works

In this exercise, you added a form to your index.php file. This is similar to the form that you added to the edit.php file. The main difference between the two is that, in this form, the `transaction_id` value is set to an empty string. No ID is necessary initially, but you want the parameter to exist so that a vehicle has been provided to pass that ID through the form when you submit the form.

Once you created the form, you added the following HTML cell definition and `<input>` element at the top of the page:

```
<td align="right" width="219" class="title">
   <input type="button" value="New Transaction" class="add" onclick="doAdd(this)">
</td>
```

As you can see, the input type is button, and it calls the JavaScript `doAdd()` function. The function links the user to the edit.php page, allowing the user to create a new transaction. At the same time, the function passes a command value of `add` to the edit.php page. That way, the edit.php page knows that a new transaction is being created and responds accordingly. You next added the following code to the initial table structure that's created in the HTML code:

```
<td width="99" class="heading"> </td>
```

This creates an additional column head in the table to provide a column for the Edit button that is added to each row returned by your query results. Finally, you added the actual cell and button to your table definition, as shown in the following code:

```
<td class="item" valign="center" align="center">
   <input type="button" value="Edit" class="edit" onclick="doEdit(this,
<?printf($transId)?>)">
</td>
```

The Edit button calls the `doEdit()` function, which passes the transaction ID to the form and links the user to the edit.php page. At the same time, the function passes the command value of `edit` so that when the edit.php page opens, it has the information it needs to allow the user to edit a current record.

Once you modified and saved the file, you opened the index.php page, created a transaction, and then edited that transaction. The application, however, still does not allow you to delete a transaction. As a result, the next section describes how you can set up PHP statements to delete data.

Deleting Data from a MySQL Database

Deleting data from the database is very similar to inserting or updating data. You must create a `DELETE` statement again and then execute the `mysql_error()` function. For example, the following PHP statements delete data from the CDs table:

```
$deleteSql = "DELETE FROM CDs WHERE CDID = 10";

mysql_query($deleteSql, $link);
   or die("Invalid query: " . mysql_error(). "<br>".$deleteSql);
```

Once again, you create a variable that holds the SQL statement. You then use that variable in the `mysql_error()` function. If necessary, you can also include a connection-related parameter in the function to ensure that a specific connection is being used. In this case, the variable is `$link`, which is similar to what you've seen in previous examples throughout the chapter. As you have also seen throughout, you can include the `or` operator and `die()` function to specify that an error be returned if the statement does not execute properly.

Deleting data is no more difficult than updating or inserting data. The key to any of these types of statements is to make sure that you set up your variables in such a way that the correct information can be passed to the SQL statement when it is being executed. In the next Try It Out section, you see how you can delete a transaction from your database. To do so, you modify the index.php file and then create a delete.php include file.

Try It Out — Modifying the index.php File and Creating the delete.php File

The following steps describe how to set up delete capabilities in your DVDRentals application:

1. First, add a column head to your table so that you can include a Delete button for each row. The button is added next to the Edit button you added in the previous Try It Out section. Add the following code (shown with the gray screen background) to your file:

```
      <tr>
         <td width="250" class="heading">Order Number</td>
         <td width="250" class="heading">Customer</td>
         <td width="250" class="heading">DVDName</td>
         <td width="185" class="heading">DateOut</td>
         <td width="185" class="heading">DateDue</td>
         <td width="185" class="heading">DateIn</td>
         <td width="99" class="heading"> </td>
         <td width="99" class="heading"> </td>
      </tr>
```

2. Next, add the code necessary to initialize variables and call the delete.php include file. Add the following code (shown with the gray screen background) to your file:

```
<?
// Connect to the server or return an error
$link = mysql_connect("localhost", "mysqlapp", "pw1")
   or die("Could not connect: " . mysql_error());

// Select the database or return an error
mysql_select_db("DVDRentals", $link)
   or die("Unable to select database: . mysql_error()");

// Initialize variables with parameters retrieved from the form
$command = null;
$transactionId = null;

if(isset($_POST["command"]))
   $command = $_POST["command"];

if(isset($_POST["transaction_id"]))
   $transactionId = $_POST["transaction_id"];

// Process the delete command
if($transactionId != null)
{
   if(strcmp("delete", $command) == 0)
   {
      include "delete.php";
   }
}
```

3. Now add the actual Delete button by adding the following code (shown with the gray screen background) to your file:

```
         <td class="item">
            <nobr>
            <?printf($dateInPrint);?>
            </nobr>
         </td>
         <td class="item" valign="center" align="center">
            <input type="button" value="Edit" class="edit" onclick="doEdit(this,
<?printf($transId)?>)">
         </td>
```

```
          <td class="item" valign="center" align="center">
             <input type="button" value="Delete" class="delete"
onclick="doDelete(this, <?printf($transId)?>)">
          </td>
       </tr>
```

4. Save the index.php file.

5. Create a new file named delete.php in your text editor, and enter the following code:

```
<?
// Build the DELETE statement
$deleteSql = "DELETE FROM Transactions WHERE TransID = '$transactionId'";

// Execute the DELETE statement
mysql_query($deleteSql)
   or die("Error in query $deleteSql " . mysql_error());
?>
```

6. Save the delete.php file to the appropriate Web application directory.

7. Open your browser, and go to the address `http://localhost/index.php`. Your browser should display a page similar to the one shown in Figure 17-5.

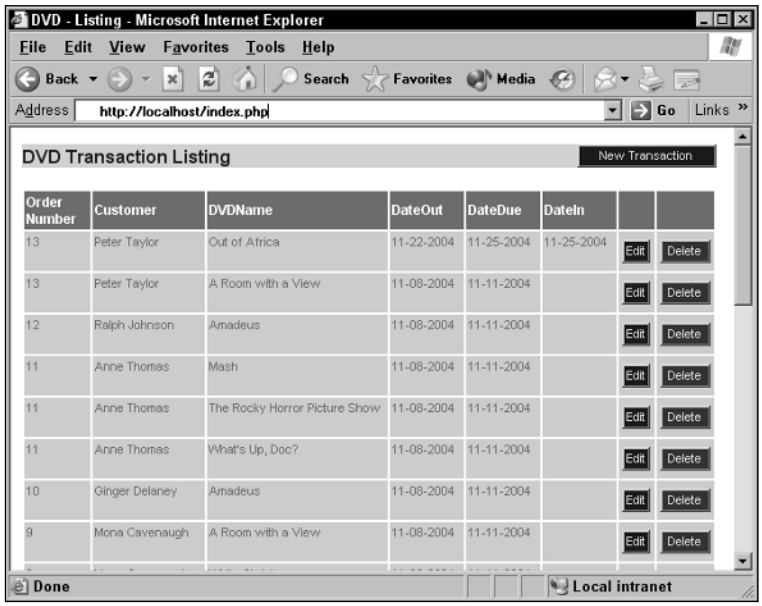

Figure 17-5

8. Click the Delete button in the row that contains the transaction that you created in a previous Try It Out section (Order number 13, DVD name *Out of Africa*). A message box similar to the one in Figure 17-6 appears, confirming whether you want to delete the record.

Figure 17-6

9. Click OK to delete the record. The index.php file should be redisplayed, with the deleted file no longer showing.

How It Works

In this exercise, you first created an additional column head for a column that holds the Delete button for each row. You then entered the following code:

```
$command = null;
$transactionId = null;

if(isset($_POST["command"]))
   $command = $_POST["command"];

if(isset($_POST["transaction_id"]))
   $transactionId = $_POST["transaction_id"];
```

First, you initialized the `$command` and `$transactionId` variables to null to ensure that they did not contain values from any previous transactions. You then used `if` statements and the `$_POST` array to assign the most current `command` and `transaction_id` variables to the variable. This process is necessary because, when you click the Delete button, you must have a mechanism in place to store the values that are submitted to the form when the button is clicked. The `$_POST` array contains the values that are submitted when the applicable JavaScript function submits the form to the index.php page.

Once you added the code necessary to initialize the variables, you added the following if statement to the file:

```
if($transactionId != null)
{
   if(strcmp("delete", $command) == 0)
   {
      include "delete.php";
   }
}
```

The `if` statement condition specifies that the `$transactionId` variable cannot be null, which it will not be if a form was submitted when the Delete button was clicked. If the condition evaluates to true, the PHP script in the delete.php include file is executed.

The last code that you added to the index.php file is the HTML cell definition and `<input>` element necessary to add the Delete button to each row displayed on the page:

```
<td class="item" valign="center" align="center">
   <input type="button" value="Delete" class="delete" onclick="doDelete(this,
<?printf($transId)?>)">
</td>
```

As you can see, the input type is button (`type="button"`), the button is named Delete (`value="Delete"`), the style is delete (`class="delete"`), and the `doDelete()` function is executed when the button is clicked. The `doDelete()` function takes two parameters. The `this` parameter merely indicates that it is the current button that is being referenced. The second parameter passes the value in the `$transID` variable to the `transactionId` parameter associated with the form. That way, PHP knows what record to delete when the `DELETE` statement is executed.

Once you set up your index.php file to support deletions, you created a delete.php file that contained the statements that actually deleted the transaction:

```
$deleteSql = "DELETE FROM Transactions WHERE TransID = '$transactionId'";
mysql_query($deleteSql)
   or die("Error in query $deleteSql " . mysql_error());
```

By now, you should be familiar with this process. The `$deleteSql` variable is set to the `DELETE` statement. The `DELETE` statement uses the `$transactionId` variable to determine which transaction to delete. The `mysql_query()` function sends the `DELETE` statement to the MySQL server. If the statement fails, the user receives an error. When the user clicks the Delete button, a pop-up message appears, verifying that the row should be deleted. If the user clicks OK, the row is deleted.

Now your application, at least this part of the application, should be complete. You can now view, insert, update, and delete transactions. In addition, you can build on this application if you want to extend your application's functionality. The code used to create this application is available at `www.wrox.com`, and you can use and modify that code as necessary. Keep in mind, however, that at the heart of your application is the MySQL database that manages the data that you need to run your application. The better you understand MySQL and data management, the more effective your applications can be.

Summary

In this chapter, you learned how you can build a PHP application that interacts with a MySQL database. The chapter first introduced you to PHP and then provided an overview of the PHP elements that you need to use to connect to your database and retrieve and manipulate data. The chapter also provided you with elements of PHP that demonstrate how PHP statements can be used to take different actions by first testing for specific conditions and then taking actions based on whether those conditions are true or false. Specifically, the chapter covered the following topics:

- Connecting to the MySQL server and selecting a database
- Using `if` statements to test conditions and take actions
- Retrieving data, formatting data, and then displaying data
- Inserting data in your database
- Updating existing data in your database
- Deleting data from your database

Much of this chapter was devoted to building an application that accessed the DVDRentals database. The application focused on retrieving and modifying data related to the transactions stored in the database. In the real world, this would no doubt represent only a part of a larger application that supports numerous other functions. The purpose of this application is merely to demonstrate the various aspects of PHP that allow you to interact with a MySQL database. The more you work with PHP and MySQL, the more you discover that the principles described in this chapter can be applied to additional functionality that allows you to build applications that are far more robust than the one shown here, while still using what you learned this chapter. Now that you have a sense of how PHP interacts with MySQL, you're ready to move on to Chapter 18, which describes how to create a Java Web-based application that uses a MySQL database.

Exercises

In this chapter, you learned how to connect to a MySQL database, retrieve data, and manipulate data from a PHP application. To assist you in better understanding how to perform these tasks, the chapter includes the following exercises. To view solutions to these exercises, see Appendix A.

1. You are establishing a connection to a MySQL server on a host named server1. To support your application, you have created a MySQL user account with the name app_user. The password for the account is app_pw. You want the connection parameters to be assigned to a variable named `$myLink`. What PHP statement should you use to establish the connection?
2. You are selecting the MySQL database for your PHP application. The name of the database is Sales_db, and your access to the database relies on the current server connection. What PHP statement should you use to select the database?
3. You plan to retrieve data from a MySQL database. You create a `SELECT` statement and assign it to the `$myQuery` variable. You want to assign the results returned by the `SELECT` statement to the `$myResult` variable. In addition, if no results are returned, you want to return an error message. The error message should begin with "Error in query:" and end with the actual error returned by MySQL. What PHP statement should you use?
4. You want to process the query results assigned to the `$myResult` variable. The query results include data from the CDName and InStock columns of the CDs table. The CDName values should be assigned to the `$cdName` variable, and the InStock values should be assigned to the `$inStock` variable. The variable values should then be printed out on your page. How should you set up your PHP code to display the returned values?
5. Your application will post data to a form that contains the user1 parameter. You want to retrieve the user1 value from within your PHP code and assign that value to the `$user1` variable. You want to assign a value to the `$user1` variable only if the user1 parameter contains a value. What PHP statement should you use?
6. You want users to be redirected to the index.php file at a certain point in your application. What PHP statement should you use to redirect users?
7. You must close the connection to the MySQL server that has been established earlier in your application. The connection parameters are stored in the `$myLink` variable. What PHP statement should you use to close the connection?