

# 13

## Putting It All Together

The 12 chapters to this point have covered all the fundamentals of using PHP for Web development. In this chapter, you'll use your accumulated knowledge to create a complete and functional Web site. And even though the focus of this chapter is on applying your newfound knowledge, you'll still learn a few new tricks. In particular, you'll see how to develop a full-scale Web application from scratch.

---

### In This Chapter

Getting Started	380
Connecting to the Database	382
Writing the User-Defined Function	383
Creating the Template	385
Logging In	388
Logging Out	392
Adding Quotes	393
Listing Quotes	397
Editing Quotes	400
Deleting Quotes	406
Creating the Home Page	410
Review and Pursue	414

---

# Getting Started

The first step when starting any project is identifying the site's goals. The primary goal of this chapter is to apply everything taught in the book thus far (a lofty aim, to be sure). The example I came up with for doing so combines the models from the previous two chapters, creating a site that will store and display quotations. Instead of using a file to do so, as in Chapter 11, "Files and Directories," this site will use a MySQL database as the storage repository. But as with the blog example from Chapter 12, "Intro to Databases," the ability to create, edit, and delete quotations will be implemented. Further, the public user will be able to view the most recent quotation by default **A**, or a random one, or a random quotation previously marked as a favorite. For improved security, the site will have an administrator who can log in and log out. And only the logged-in administrator will be allowed to create, edit, or delete quotations **B**. The site will use a simple template to give every page a consistent look, with CSS handling all the formatting and layout. The site will also make use of one user-defined function, stored in an included file.

As in Chapter 12, you must first create the database and its one table. The database could be named *myquotes* (or something else if you'd rather). You create its one table with this SQL command:

```
CREATE TABLE quotes (  
  quote_id INT UNSIGNED NOT NULL  
  → AUTO_INCREMENT,  
  quote TEXT NOT NULL,  
  source VARCHAR(100) NOT NULL,  
  favorite TINYINT(1) UNSIGNED NOT  
  → NULL,  
  date_entered TIMESTAMP NOT NULL  
  → DEFAULT CURRENT_TIMESTAMP,  
  PRIMARY KEY (quote_id)  
)
```



**A** The site's simple home page.



**B** Nonadministrators are denied access to certain pages.

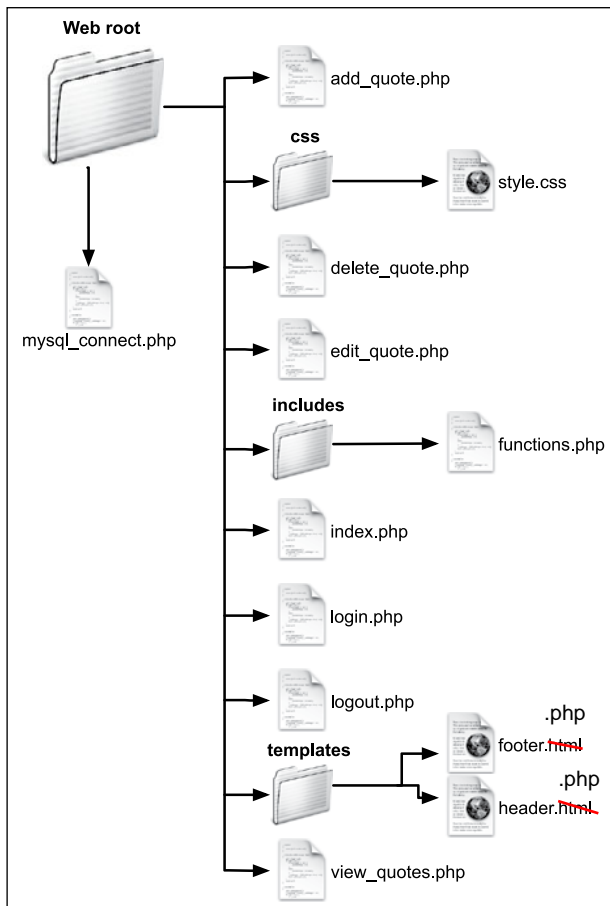
The **quote\_id** is the primary key, and it will automatically be incremented to the next logical value with each new submission. The **quote** field stores the quote itself, with the **source** field storing the attribution (unlike in Chapter 11, where the quote and the source were stored together). The **favorite** field stores a 1 or a 0, marking the quote as a favorite or not. Finally, the **date\_entered** column is a timestamp, automatically set to the current timestamp when a new record is created.

You can create this table using a PHP script, such as Script 12.4, or a third-party

application (like the MySQL client or phpMyAdmin).

Finally, a word about how the site should be organized on the server **C**. Ideally the **mysql\_connect.php** script, which establishes a database connection, would be stored outside the Web root directory. If that is not possible in your case, you can place it in the **includes** folder, then change the code in the other scripts accordingly.

Once you've created the database, the database table, and the necessary folders, you can begin coding.



**C** The structure and organization of the files on the server.

REMEMBER: We are using mysql

## Connecting to the Database

Unlike the scripts in Chapter 12, which connected to the database using code repeated in each script, this site will use the more common practice of placing the repeated code in a stand-alone file. Every script that interacts with the database—which will be most but not all of them—will then include this file. As you can see in Figure 13.1 in the previous section, this file should be stored outside of the Web root directory or, if that's not possible, within the **includes** folder.

### To create `mysql_connect.php`:

1. Begin a new PHP script in your text editor or IDE, to be named **`mysql_connect.php`** (Script 13.1):  

```
<?php
```
2. Connect to the database server:  

```
$dbc = mysqli_connect('localhost',  
'username', 'password','database');
```

Naturally, you'll need to change the values used here to be appropriate for your server.
3. Select the database:  

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

If you named your database something else, change this value accordingly.
4. Complete the script:  

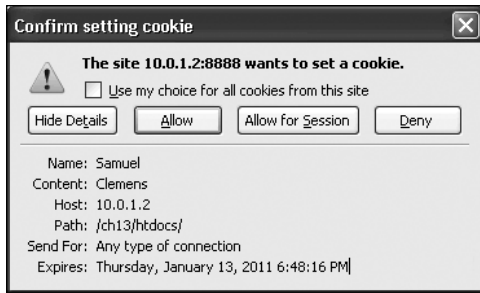
```
?>
```
5. Save the file as **`mysql_connect.php`**.

**Script 13.1** This script connects to the database server and selects the database to be used.

```
1 <?php // Script 13.1 - mysql_connect.php
2 /* This script connects to and selects
   the database. */
3
4 // Connect:
5 $dbc = mysqli_connect('localhost',
6 'username', 'password','database');
7
8 // Select:
9 mysql_select_db('myquotes', $dbc);
10 ?>
```

```
DEFINE ('DB_HOST', 'localhost');
DEFINE ('DB_USER', 'phpstuXX');
DEFINE ('DB_PASSWORD', 'XXXXX');
DEFINE ('DB_NAME', 'my_quotesXX'); //this is
unique to your classroom student ID
//DEFINE ('DB_HOST', 'localhost');
//DEFINE ('DB_USER', 'root');
//DEFINE ('DB_PASSWORD', 'root');
//DEFINE ('DB_NAME', 'myquotes');
//Make the connection:
if ($dbc = @mysqli_connect(DB_HOST,
DB_USER, DB_PASSWORD, DB_NAME) ) {
echo '<p style="color:red;">You have
successfully connected to MySQL.';

//mysqli_close($dbc); //Comment this out after
you do a DB connection. It will screw up the
rest of the files.
} else {
echo '<p style="color:red;">' . die('Could not
connect to MySQL: '
. mysqli_connect_error() ) . '</p>';
} //die function terminates the execution of the
script
```



**A** This cookie will be used to identify the administrator.

## Writing the User-Defined Function

The site will have a single user-defined function. As discussed in Chapter 10, “Creating Functions,” the best time to create your own functions is when a script or a site might have repeating code. In this site there are a couple such instances, but the most obvious one is this: Many scripts will need to check whether or not the current user is an administrator. In this next script, you’ll create your own function that returns a Boolean value indicating if the user is an administrator. But what will be the test of administrative status?

Upon successfully logging in, a cookie will be sent to the administrator’s browser, with a name of *Samuel* and a value of *Clemens*. **A** This may seem odd or random, and it is. When using something simple, like a cookie, for authentication, it’s best to be obscure about what constitutes verification. If you went with something more obvious, such as a name of *admin* and a value of *true*, that’d be quite easy for anyone to guess and falsify. With this cookie in mind, this next function simply checks if a cookie exists with a name of *Samuel* and a value of *Clemens*.

# Creating and Calling Functions That Take Arguments

The exact formatting of a function isn't important as long as the requisite elements are there. These elements include the word *function*, the function's name, the opening and closing parentheses, the opening and closing brackets, and the statement(s). It's conventional to indent a function's statement(s) from the **function** keyword line, for clarity's sake, as you would with a loop or conditional. In any case, select a format style that you like (which is both syntactically correct and logically sound) and stick to it.

You call (or invoke) the function by referring to it just as you do any built-in function. The line of code

```
whatever();
```

will cause the statement part of the previously defined function—the **print** command—to be executed.

Although being able to create a simple function is useful, writing one that takes input and does something with that input is even better. **The input a function takes is called an *argument* (or a *parameter*).**

This is a concept you've seen before: the **sort()** function takes an array as an argument, which the function then sorts.

The syntax for writing functions that take arguments is as follows:

```
function function_name($arg1,  
→ $arg2, ...){  
    statement(s);  
}
```

The function's arguments are in the form of variables that are assigned the values sent to the function when you call it. The variables are defined using the same naming rules as any other variable in PHP:

```
function make_full_name($first,  
→ $last) {  
    print $first . ' ' . $last;  
}
```

Functions that take input are called much like those that don't—you just need to remember to pass along the necessary values. You can do this either by passing variables:

```
make_full_name($fn, $ln);
```

or by sending literal values, as in

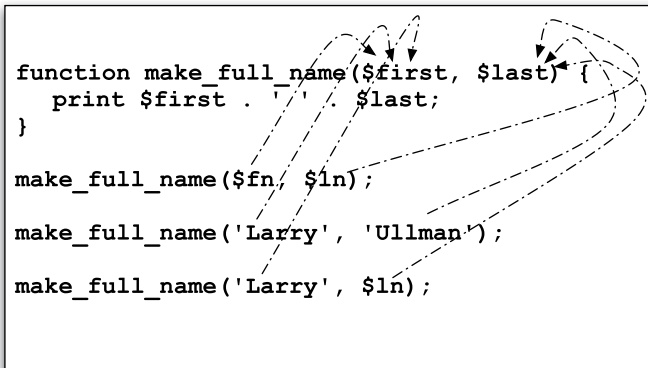
```
make_full_name('Larry', 'Ullman');
```

or some combination thereof:

```
make_full_name('Larry', $ln);
```

*continues on next page*

The important thing to note is that arguments are passed quite literally: The first variable in the function definition is assigned the first value in the call line, the second function variable is assigned the second call value, and so forth **A**. Functions aren't smart enough to intuitively understand how you meant the values to be associated. This is also true if you fail to pass a value, in which case the function will assume that value is null (*null* isn't the mathematical 0, which is actually a value, but closer to the idea of the word *nothing*). The same thing applies if a function takes four arguments and you pass three—the fourth will be null, which may create an error **B**.



The diagram illustrates how arguments are passed to a function. It shows a function definition `function make_full_name($first, $last) { print $first . ' ' . $last; }` and three function calls: `make_full_name($fn, $ln);`, `make_full_name('Larry', 'Ullman');`, and `make_full_name('Larry', $ln);`. Dashed arrows connect the arguments in each call to the corresponding parameters in the function definition. For the first call, `$fn` points to `$first` and `$ln` points to `$last`. For the second call, the string `'Larry'` points to `$first` and `'Ullman'` points to `$last`. For the third call, `'Larry'` points to `$first` and `$ln` points to `$last`.

```
function make_full_name($first, $last) {  
    print $first . ' ' . $last;  
}  
  
make_full_name($fn, $ln);  
  
make_full_name('Larry', 'Ullman');  
  
make_full_name('Larry', $ln);
```

**A** How values in function calls are assigned to function arguments.

**Warning:** Missing argument 2 for `make_text_input()`, called in `/Users/larryullman/Sites/phpvqs4/sticky1.php` on line 38 and defined in `/Users/larryullman/Sites/phpvqs4/sticky1.php` on line 14

**B** As with any function (user-defined or built into PHP), passing an incorrect number of arguments when calling it yields warnings.

## To create functions.php:

1. Begin a new PHP script in your text editor or IDE, to be named **functions.php** (Script 13.2):

```
<?php // Script 13.2 - functions.php
```

Even though, as written, the script only defines a single function, I'm naming the file using the plural—*functions*—with the understanding that more user-defined functions might be added to the script in time.

2. Begin defining a new function:

```
function is_administrator($name =  
→ 'Samuel', $value = 'Clemens') {
```

The function takes two arguments: the cookie's name and its value. Both have default values.

Since the function checks only a single cookie with a single expected value, it doesn't need to take arguments at all, let alone default ones. But by having the function take arguments, it can be used in different ways should the

site's functionality expand (e.g., if you had multiple types of authentication to perform). Still, by using default values for those arguments, function calls don't need to provide argument values when the assumed cookie name and value are being checked.

3. Return a Boolean value based on the cookie's existence and value:

```
if (isset($_COOKIE[$name]) &&  
→ ($_COOKIE[$name] == $value)) {  
    return true;  
} else {  
    return false;  
}
```

If the cookie exists and has the appropriate value, the Boolean true is returned. Otherwise the function returns false.

4. Complete the function and the script:

```
} // End of is_administrator()  
→ function.  
?>
```

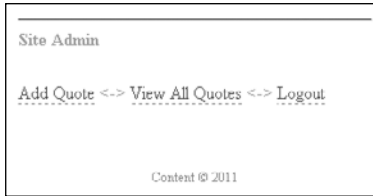
5. Save the file as **functions.php**, stored in the **includes** directory.

**Script 13.2** The `is_administrator()` function, defined in an includable script, will be called on any page that needs to verify administrator status.

```
1  <?php // Script 13.2 - functions.php  
2  /* This page defines custom functions. */  
3  
4  // This function checks if the user is an administrator.  
5  // This function takes two optional values.  
6  // This function returns a Boolean value.  
7  function is_administrator($name = 'Samuel', $value = 'Clemens') {  
8  
9      // Check for the cookie and check its value:  
10     if (isset($_COOKIE[$name]) && ($_COOKIE[$name] == $value)) {  
11         return true;  
12     } else {  
13         return false;  
14     }  
15  
16 } // End of is_administrator() function.  
17  
18 ?>
```

When you're creating functions, it's easy to create parse errors by forgetting the closing curly bracket. You may want to add comments to help you remember this final step.





**A** If the person viewing any page is an administrator, they'll see some additional links.

**Script 13.3** The header file includes the **functions.php** script and begins the HTML page.

```
1  <?php // Script 13.3 - header.php
2
3  // Include the functions script:
4  include('includes/functions.php'); ?>
5  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
6    1.0 Transitional//EN"
7    "http://www.w3.org/TR/xhtml1/DTD/
8    xhtml1-transitional.dtd">
9  <html xmlns="http://www.w3.org/1999/
10     xhtml" xml:lang="en" lang="en">
11  <head>
12    <meta http-equiv="content-type"
13      content="text/html; charset=utf-8" />
14    <link rel="stylesheet" media="all"
15      href="css/style.css" />
16    <title><?php // Print the page title.
17    if (defined('TITLE')) { // Is the title
18      defined?
19        print TITLE;
20      } else { // The title is not defined.
21        print 'My Site of Quotes';
22      }
23    }></title>
24  </head>
25  <body>
26    <div id="container">
27      <h1>My Site of Quotes</h1>
28      <br />
29      <!-- BEGIN CHANGEABLE CONTENT. -->
```

## Creating the Template

Now that the two helper files have been created, it's time to move on to the template. As illustrated in Chapter 8, "Creating Web Applications," the site's layout will be controlled by two includable files: a header and a footer. Both will be stored in the **templates** directory. The header also references a style sheet, to be stored in the **css** folder. You can find the style sheet by downloading the book's code from [www.LarryUllman.com](http://www.LarryUllman.com) (the file will be in the **ch13** folder of the download).

Besides generating the primary HTML for the site, the header file must include the functions script. The footer file should also display some administration links **A**, should the current user be an administrator.

### To create header.html:

1. Begin a new HTML document in your text editor or IDE, to be named **header.php** (Script 13.3).

```
<?php // Script 13.3 - header.php
include('includes/functions.php'); ?>
```

The header will start with a PHP section in order to include the **functions.php** script, which will be required by multiple pages on the site. The reference to the script is relative to the pages that will be including the header: files in the main directory.

*continues on next page*

2. Begin the HTML document:

```
<!DOCTYPE html>

<html>
<head>

    <meta charset="utf-8">
<link rel="stylesheet" media=
→ "all" href="css/style.css" /
>
```

Again, the style sheet needs to be downloaded from the book's corresponding Web site. And the reference to that style sheet is relative to the scripts that include the header, all found within the main directory.

3. Print the page's title:

```
<title><?php
if (defined('TITLE')) {
    print TITLE;
} else {
    print 'My Site of Quotes';
}
?></title>
```

This code also comes from Chapter 8, printing either the default title or a custom one, if set as a constant.

4. Complete the head:

```
</head>
```

5. Begin the page's body:

```
<body>
    <div id="container">
        <h1>My Site of Quotes</h1>
        <br />
        <!-- BEGIN CHANGEABLE
→ CONTENT. -->
```

6. Save the file as **header.php**, stored in the **includes** directory.

## To create footer.html:

1. Begin a new HTML document in your text editor or IDE, to be named **footer.php** (Script 13.4):  

```
<!-- END CHANGEABLE CONTENT. -->
```
2. Check if it's appropriate to display general administration links:

```
<?php
if ( (is_administrator() &&
→ (basename($_SERVER['PHP_SELF'])
→ != 'logout.php'))
OR (isset($loggedin) &&
→ $loggedin) ) {
```

This conditional is a bit complicated. To start, most pages can confirm that the current user is an administrator by just invoking the **is\_administrator()** function. But because of how cookies work, that function will return inappropriate results on two pages: **login.php** and **logout.php**. On the **logout.php**, the script will have received the administrative cookie prior to deleting it. So on that page, it will seem like the user is an administrator (because they just were), but links to the administrative pages should not be shown as the user will be blocked from using them (because when they get to those pages, the cookie will no longer exist). Hence, the first part of the conditional requires that **is\_administrator()** return true and that the current page not be **logout.php**. The code **basename(\$\_SERVER['PHP\_SELF'])** is a reliable way to get the current script (and because the footer file is included by another script, **\$\_SERVER['PHP\_SELF']** will have the value of the *including* script.

The second part of the conditional, after the **OR**, checks if the **\$loggedin** variable is set and has a true value. This will be the case on the **login.php** page, after the user successfully logged in. The

`is_administrator()` function won't return a true value at that juncture, because the cookie will have been just sent by the script, and therefore won't be available to be read.

3. Create the links:

```
print '<hr><h3>Site Admin</h3>' →
<p><a href="add_quote.php">Add →
Quote</a> <->
<a href="view_quotes.php">View All
→ Quotes</a> <->
<a href="logout.php">Logout</a></p>;
```

Three links are created: to a page for adding new quotes, to a page for viewing every quote, and to the logout page.

4. Complete the conditional and the PHP section:

```
}
?>
```

5. Complete the HTML page:

```
</div><!-- container -->
<div id="footer">Content
&copy; 2014</div>
</body> SEE NOTE BELOW
</html>
```

6. Save the file as `footer.php`, stored in the `includes` directory.

**Script 13.4** The footer file displays general administrative links, when appropriate, and completes the HTML page.

```
1  <!-- END CHANGEABLE CONTENT. -->
2  <?php // Script 13.4 - footer.php
3
4  // Display general admin links...
5  // - if the user is an administrator and it's not the logout.php page
6  // - or if the $loggedin variable is true (i.e., the user just logged in)
7  if ( (is_administrator() && (basename($_SERVER['PHP_SELF']) != 'logout.php'))
8  OR (isset($loggedin) && $loggedin) ) {
9
10     // Create the links:
11     print '<hr /><h3>Site Admin</h3><p><a href="add_quote.php">Add Quote</a> <->
12     <a href="view_quotes.php">View All Quotes</a> <->
13     <a href="logout.php">Logout</a></p>;
14
15 }
16
17 ?>
18 </div><!-- container -->
19 <div id="footer">Content &copy; 2014</div>
20 </body>
21 </html>
```

```
<div id="footer">
<p>&copy;
<?php
$startYear = 2014;
$thisYear = date('Y');
if ($startYear == $thisYear) {
    echo $startYear . ' ';
} else {
    echo "{$startYear}&#8211;{$thisYear}";
}
?>
&#124; Your Name Goes Here</p>
```

## Logging In

Next, it's time to create the script through which the administrator can log in. The end result will be very similar to the scripts in Chapter 9, "Cookies and Sessions," with one structural difference: that chapter used *output buffering*, allowing you to lay out your script however you want. This site does not use output buffering, so the handling of the form must be written in such a way that the cookie can be sent without generating *headers already sent* errors (see Chapter 9 if this isn't ringing a bell for you). In other words, the script must check for a form submission prior to including the header. To still be able to reflect errors and other messages within the context of the page, you must therefore use variables.

### To create login.php:

1. Begin a new PHP document in your text editor or IDE, to be named **login.php** (Script 13.5):

```
<?php // Script 13.5 - login.php
```

2. Define two variables with default values:

```
$loggedin = false;
$error = false;
```

These two variables will be used later in the script. Here they are given default values, indicating that the person is not logged in and no errors have yet occurred.

3. Check if the form has been submitted:

```
if ($_SERVER['REQUEST_METHOD'] ==
    'POST') {
```

**Script 13.5** The login script both displays and handles a form, sending a cookie upon a successful login attempt.

```
1  <?php // Script 13.5 - login.php
2  /* This page lets people log into the
   site. */
3
4  // Set two variables with default values:
5  $loggedin = false;
6  $error = false;
7
8  // Check if the form has been submitted:
9  if ($_SERVER['REQUEST_METHOD'] == 'POST')
10 {
11     // Handle the form:
12     if (!empty($_POST['email']) &&
        !empty($_POST['password'])) {
13
14         if ( ( strtolower($_POST['email'])
15             == 'me@example.com') && ($_
16                 POST['password'] == 'testpass') ) {
17             // Correct!
18
19             // Create the cookie:
20             setcookie('Samuel', 'Clemens',
21                 time()+3600);
22
23             // Indicate they are logged in:
24             $loggedin = true;
25
26         } else { // Incorrect!
27
28             $error = 'The submitted email
29                 address and password do not
30                 match those on file!';
31
32         }
33     } else { // Forgot a field.
34
35         $error = 'Please make sure you
36             enter both an email address and a
37             password!';
38     }
39 }
```

code continues on next page

Cookies are created when a user's [browser](#) loads a particular website. The website sends information to the browser which then creates a text file. Every time the user goes back to the same website, the browser retrieves and sends this file to the website's server. Computer Cookies are created not just by the website the user is browsing but also by other websites that run ads, widgets, or other elements on the page being loaded. These cookies regulate how the ads appear or how the widgets and other elements function on the page. <http://www.allaboutcookies.org/>

### Script 13.5 continued

```
36 // Set the page title and include the
    header file:
37 define('TITLE', 'Login');
38 include('templates/header.html');
39
40 // Print an error if one exists:
41 if ($error) {
42     print '<p class="error">' . $error .
    '</p>';
43 }
44
45 // Indicate the user is logged in, or
    show the form:
46 if ($loggedin) {
47
48     print '<p>You are now logged in!</p>';
49
50 } else {
51
52     print '<h2>Login Form</h2>
53     <form action="login.php"
54     method="post">
55     <p><label>Email Address <input type=
56     "email" name="email"></label></p>
57     <p><label>Password <input
58     type="password" name="password">
59     </label></p>
60     <p><input type="submit"
61     name="submit" value="Log In!"></p>
62     </form>';
63
64 }
65
66 include('templates/footer.php');
    // Need the footer.
67 ?>
```

### 4. Handle the form:

```
if (!empty($_POST['email']) &&
→ !empty($_POST['password'])) {
    if ( ( strtolower($_POST['email'])
→ == 'me@example.com') && ($_POST
→ ['password'] == 'testpass') ) {
```

Similar to examples in Chapter 9, this script first confirms that both `$_POST['email']` and `$_POST['password']` are not empty. The second conditional compares the submitted values against what they need to be.

### 5. Create the cookie:

```
setcookie('Samuel', 'Clemens',
→ time()+3600);
```

The cookie has a name of *Samuel* and a value of *Clemens*. It's set to expire in an hour.

### 6. Indicate that the user is logged in:

```
$loggedin = true;
```

This variable will be used later in this same script, and in the footer file (see Script 13.4).

### 7. Create error messages for the two other conditions:

```
    } else { // Incorrect!
        $error = 'The submitted
        → email address and
        → password do not match
        → those on file!';
    }
    } else { // Forgot a field.
        $error = 'Please make sure
        → you enter both an email
        → address and a password!';
    }
}
```

*continues on next page*

The first **else** clause applies if an email address and password were provided but were wrong. The second **else** clause applies if no email address or password was submitted. In both cases, the message is assigned to a variable so that it may be used later in the script.

8. Set the page title and include the header file:

```
define('TITLE', 'Login');  
include('templates/header.php');
```

9. Print an error if one exists:

```
if ($error) {  
    print '<p class="error">' .  
        → $error . '</p>';  
}
```

The error itself will be determined in Step 7, but it can't be printed at that point because the HTML header will not have been included. The solution is to have this code check for a non-false **\$error** value and then print **\$error** within some HTML and CSS **A**.

## My Site of Quotes

The submitted email address and password do not match those on file!

### Login Form

Email Address

**A** Error messages are displayed after the header is included but before the HTML form.



**B** The result upon successfully logging in.



**C** The basic login form.

10. Indicate that the user is logged in, or display the form:

```
if ($loggedin) {
    print '<p>You are now logged
    → in!</p>';
} else {
    print '<h2>Login Form</h2>
    <form action="login.php"
    → method="post">
    <p><label>Email Address <input
    → type="email" name="email">
    → </label></p>
    <p><label>Password <input
    → type="password" name=
    → "password"></label></p>
    <p><input type="submit" name=
    → "submit" value="Log In!"></p> </
    form>';
}
```

If the **\$loggedin** variable has a true value—its value is false by default—then the user just successfully logged into the site and a message saying so is displayed **B**. If the **\$loggedin** variable still has its default value, then the form should be shown **C**.

11. Include the footer and complete the page:  
`include('templates/footer.php'); ?>`

12. Save the file as **login.php**.

13. Test the file in your Web browser (Figures **A**, **B**, and **C**).

I purposefully omitted creating a link to the login section (as a security measure), so you'll need to enter the correct URL in your Web browser's address bar.

**TIP** Note that, as written, the script requires administrators to log back in after an hour, whether or not they continue to be active in the site.

# Logging Out

If you write a login process, there must be a logout process, too. In this case, just using cookies, it's a very simple script.

## To create `logout.php`:

1. Begin a new PHP document in your text editor or IDE, to be named **logout.php** (Script 13.6):

```
<?php // Script 13.6 - logout.php
```

2. Destroy the cookie, but only if it already exists:

```
if (isset($_COOKIE['Samuel'])) {  
    setcookie('Samuel', FALSE,  
        → time()-300);  
}
```

As an extra security measure, the script only attempts to delete the cookie if it exists. By making this check, the script prevents hackers from discovering the name of the cookie used by the site by accessing the logout script without having first logged in.

To delete the existing login cookie, another cookie is sent with the same name, a value of FALSE, and an expiration time in the past.

3. Define a page title and include the header:

```
define('TITLE', 'Logout');  
include('templates/header.html');
```

4. Print a message:

```
print '<p>You are now logged  
→ out.</p>';
```

5. Include the footer:

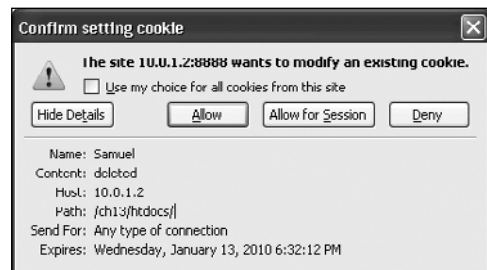
```
include('templates/footer.html');  
?>
```

6. Save the file as **logout.php**.

7. Test the file in your Web browser **A** and **B**.

**Script 13.6** The logout script deletes the administrator-identifying cookie.

```
1 <?php // Script 13.6 - logout.php  
2 /* This is the logout page. It destroys  
   the cookie. */  
3  
4 // Destroy the cookie, but only if it  
   already exists:  
5 if (isset($_COOKIE['Samuel'])) {  
6     setcookie('Samuel', FALSE,  
       time()-300);  
7 }  
8  
9 // Define a page title and include the  
   header:  
10 define('TITLE', 'Logout');  
11 include('templates/header.php');  
12  
13 // Print a message:  
14 print '<p>You are now logged out.</p>';  
15  
16 // Include the footer:  
17 include('templates/footer.php');  
18 ?>
```



- A** This cookie deletes the existing cookie.



- B** The resulting logout page.



**A** The form for adding quotations to the database.

**Script 13.7** The `add_quote.php` script allows only administrators to add new quotations to the database.

```

1  <?php // Script 13.7 - add_quote.php
2  /* This script adds a quote. */
3
4      // Define a page title and include
   the header:
5  define('TITLE', 'Add a Quote');
6  include('templates/header.php');
7
8  print '<h2>Add a Quotation</h2>';
9
10 // Restrict access to administrators only:
11 if (!is_administrator()) {
12     print '<h2>Access Denied!</h2><p
   class="error">You do not have
   permission to access this page.</p>';
13 include('templates/footer.php');
14 exit();
15 }
16
17 // Check for a form submission:
18 if ($_SERVER['REQUEST_METHOD'] == 'POST')
19 { // Handle the form.
20     if ( !empty($_POST['quote']) &&
21         !empty($_POST['source']) ) {

```

*code continues on next page*

## Adding Quotes

Now that the administrator has the ability to log in, she or he should be able to start adding quotations. The script for doing so is a lot like the one for adding blog postings (from Chapter 12), but the form will have an additional checkbox for marking the quotation as a favorite **A**.

Because this script creates records in the database, security must be a primary concern. As an initial precaution, the script will make sure that only the administrator can use the page. Second, values to be used in the SQL command will be sanctified to prevent invalid queries.

### To create `add_quote.php`:

1. Begin a new PHP document in your text editor or IDE, to be named `add_quote.php` (Script 13.7):
 

```

<?php // Script 13.7 -
→ add_quote.php
define('TITLE', 'Add a Quote');
include('templates/header.php');
print '<h2>Add a Quotation</h2>';

```
2. Deny access to the page if the user is not an administrator:
 

```

if (!is_administrator()) {
    print '<h2>Access Denied!</h2>
→ <p class="error">You do not
→ have permission to access
→ this page.</p>';
    include('templates/footer.php');
    exit();
}

```

*continues on next page*

By invoking the `is_administrator()` function, the script can quickly test for that condition. If the user is not an administrator, an *Access Denied* error is displayed, the footer is included, and the script is terminated **B**.

**3.** Check for a form submission:

```
if ($_SERVER['REQUEST_METHOD'] ==  
→ 'POST') {
```

**4.** Check for values:

```
if ( !empty($_POST['quote']) &&  
→ !empty($_POST['source']) ) {
```

This script performs a minimum of validation, checking that the two variables aren't empty. With large blocks of text, such as a quotation, there's not much more that can be done in terms of validation.

**5.** Prepare the values for use in the query:

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

To make the two textual values safe to use in the query, they're run through the `mysqli_real_escape_string()` function (see Chapter 12). To make the values safe to later display in the Web page, the `strip_tags()` function is applied, too (see Chapter 5, "Using Strings").

**6.** Create the *favorite* value:

```
if (isset($_POST['favorite'])) {  
    $favorite = 1;  
} else {  
    $favorite = 0;  
}
```

## My Site of Quotes

Add a Quotation  
Access Denied!

You do not have permission to access this page.

Content © 2011

**B** Any user not logged in as an administrator is denied access to the form.

**Script 13.7** *continued*

```
22         // Need the database connection:  
23         include('../mysqli_connect.php');  
24  
25         // Prepare the values for storing:  
26         $quote = mysqli_real_escape_  
            string(trim(strip_tags($_  
                POST['quote'])), $dbc);  
27         $source = mysqli_real_escape_  
            string(trim(strip_tags($_  
                POST['source'])), $dbc);  
28  
29         // Create the "favorite" value:  
30         if (isset($_POST['favorite'])) {  
31             $favorite = 1;  
32         } else {  
33             $favorite = 0;  
34         }  
35  
36         $query = "INSERT INTO quotes  
            (quote, source, favorite) VALUES  
            ('$quote', '$source', '$favorite')";  
37         $r = mysqli_query($dbc, $query);  
38  
39         if (mysqli_affected_rows($dbc) == 1)  
            {  
40             // Print a message:  
41             print '<p>Your quotation has  
                been stored.</p>';  
42         } else {  
43             print '<p class="error">Could not  
                store the quote because:<br />' .  
                mysqli_error($dbc) . '</p><p>The  
                query being run was: ' . $query .  
                '</p>';
```

*code continues on next page*

**Script 13.7** *continued*

```

44     }
45
46     // Close the connection:
47     mysqli_close($dbc);
48
49     } else { // Failed to enter a quotation.
50         print ' <p class="error">Please
           enter a quotation and a source!</p>';
51     }
52
53 } // End of submitted IF.
54
55 // Leave PHP and display the form:
56 ?>
57
58 <form action="add_quote.php"
   method="post">
59     <p><label>Quote <textarea
       name="quote" rows="5" cols="30">
       </textarea></label></p>
60     <p><label>Source <input type="text"
       name="source"></label></p>
61     <p><label>Is this a favorite? <input
       type="checkbox" name="favorite"
       value="yes"></label></p>
62     <p><input type="submit" name="submit"
       value="Add This Quote!"></p>
63 </form>
64
65 <?php include('includes/header.php'); ?>

```

In the database, a quotation's status as a favorite is indicated by a 1. Nonfavorites are represented by a 0. To determine which number to use, all the PHP script has to do is check for a **\$\_POST ['favorite']** value. If that variable is set, regardless of what its value is, it means the user checked the favorite checkbox. If the user didn't do that, then the variable won't be set, and the **\$favorite** variable will be assigned the value 0.

**7.** Define and execute the query:

```

$query = "INSERT INTO quotes
→ (quote, source, favorite) VALUES
→ ('$quote', '$source', $favorite)";
$r = mysqli_query($query, $dbc);

```

The query specifies values for three fields and uses the variables already defined. The remaining two table columns—**quote\_id** and **date\_entered**—will automatically be assigned values, thanks to the table's definition.

**8.** Print a message based on the results:

```

if (mysqli_affected_rows($dbc)
== → 1) {
    print ' <p>Your quotation has
    → been stored.</p>';
} else {
    print ' <p class="error">Could
    → not store the quote because:
    <br />' . mysqli_error($dbc) .
    → ' .</p><p>The query being run
    was: ' . $query . ' </p>';
}

```

If the **INSERT** query created one new row in the database, a message indicating such is displayed. Otherwise, debugging information is shown so that you can try to figure out what went wrong.

*continues on next page*

9. Complete the validation conditional:

```
} else { // Failed to enter a
→ quotation.
    print '<p class="error">Please
→ enter a quotation and a
→ source!</p>';
}
```

10. Complete the submission conditional and the PHP block:

```
} // End of submitted IF.
?>
```


11. Create the form:

```
<form action="add_quote.php"
→ method="post">
    <p><label>Quote <textarea
→ name="quote" rows="5"
→ cols="30"></textarea>
→ </label></p>
    <p><label>Source <input
→ type="text" name="source">
→ </label></p>
    <p><label>Is this a favorite?
→ <input type="checkbox"
→ name="favorite"></label></p>
    <p><input type="submit"
→ name="submit" value="Add This
Quote!"></p>
</form>
```


The form has one text area, one text input, and a checkbox (plus the submit button, of course). It is not designed to be sticky: that's a feature you could add later, if you wanted.

12. Include the footer:

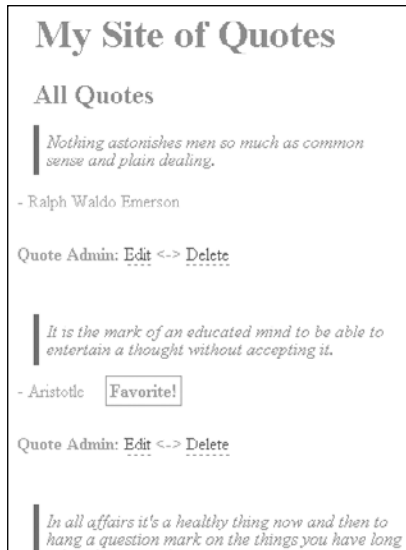
```
<?php include('includes/→
footer.php'); ?>
```

13. Save the file as **add\_quote.php** and test in your Web browser .



-  The result after adding a quotation.

**TIP** The ability to create (i.e., INSERT), retrieve (SELECT), update, and delete database records is collectively referred to as **CRUD**.



**A** The full list of stored quotations, with links to edit or delete each.

**Script 13.8** This script lists every quotation currently stored, providing links for the administrator to edit or delete them.

```

1  <?php // Script 13.8- view_quotes.php
2  /* This script lists every quote. */
3
4      // Include the header:
5  define('TITLE', 'View All Quotes');
6  include('includes/header.php');
7
8  print '<h2>All Quotes</h2>';
9
10 // Restrict access to administrators only:
11 if (!is_administrator()) {
12     print '<h2>Access Denied!</h2><p
13         class="error">You do not have
14         permission to access this page.</
15         p>'; include('includes/footer.php');
16         exit();
17     }
18 // Need the database connection:18
19 include('includes/mysqli_connect.php');
20
```

code continues on next page

## Listing Quotes

The administrative side of the site will have a page that lists every quote stored in the database **A**. Although the same script could easily be adapted for the public side, its primary purpose is to provide quick links for the administrator to edit or delete any quote (as opposed to searching randomly through the public side for the right quote to manage).

Like the `add_quote.php` script, this page will restrict access to just administrators.

### To create `view_quotes.php`:

1. Begin a new PHP document in your text editor or IDE, to be named `view_quotes.php` (Script 13.8):
 

```

<?php // Script 13.8-
→ view_quotes.php
define('TITLE', 'View All Quotes');
include('includes/header.php'); print
'<h2>All Quotes</h2>';

```
2. Terminate the script if the user isn't an administrator:
 

```

if (!is_administrator()) {
    print '<h2>Access Denied!</h2>
→ <p class="error">You do not
→ have permission to access
→ this page.</p>';
    include('includes/footer.php');
    exit();
}

```

continues on next page

This is the exact same code as that in **add\_quote.php**. Except for the browser's title, the result for nonadministrators will be the same as in **B** in the previous section of the chapter.

3. Include the database connection and define the query:

```
include('includes/mysql_connect.php');
$query = 'SELECT quote_id, quote, →
source, favorite FROM quotes
→ ORDER BY date_entered DESC';
```

```
<!-- BEGIN CHANGABLE CONTENT. --><h2>All Quo
<p><b>Quote Admin:</b> <a href="edit_quote.php?id=13">Edit</a>
<a href="delete_quote.php?id=13">Delete</a></p>
<div><blockquote>It is the mark of an educated mind to be abl
<strong>Favorite!</strong><p><b>Quote Admin:</b> <a href="ed
<a href="delete_quote.php?id=9">Delete</a></p>
<div><blockquote>In all affairs it's a healthy thing now and
mark on the things you have long taken for granted.</blockquote>
<p><b>Quote Admin:</b> <a href="edit_quote.php?id=5">Edit</a>
<a href="delete_quote.php?id=5">Delete</a></p>
<div><blockquote>Nurture your mind with great thoughts, for y
<p><b>Quote Admin:</b> <a href="edit_quote.php?id=6">Edit</a>
<a href="delete_quote.php?id=6">Delete</a></p>
```

**B** The HTML source code for the page shows how the **quote\_id** value is passed in the URL to the linked pages.

#### Script 13.8 continued

```
20 // Define the query:
21 $query = 'SELECT quote_id, quote, source, favorite FROM quotes ORDER BY date_entered DESC';
22
23 // Run the query:
24 if ($r = mysqli_query($dbc, $query)) {
25
26     // Retrieve the returned records: while
27     ($row = mysqli_fetch_array($r)) {
28
29         // Print the record:
30         print "<div><blockquote>{$row['quote']}</blockquote>- {$row['source']}\n";
31
32         // Is this a favorite?
33         if ($row['favorite'] == 1) {
34             print ' <strong>Favorite!</strong>';
35         }
36
37         // Add administrative links:
38         print "<p><b>Quote Admin:</b> <a href='\"edit_quote.php?id={$row['quote_id']}'\">Edit</a> <->
39         <a href='\"delete_quote.php?id={$row['quote_id']}'\">Delete</a></p></div>\n";
40
41     } // End of while loop.
42
43 } else { // Query didn't run.
44     print '<p class="error">Could not retrieve the data because:<br>' . mysqli_error($dbc) . ' .</p>';
45     print "<p>The query being run was: ' . $query . '</p>";
46 } // End of query IF.
47
48 mysqli_close($dbc); // Close the connection.
49
50 include('includes/footer.php'); // Include the footer.
51 ?>
```

The query returns four columns—all but the date entered—from the database for every record. The results will be returned in order by the date they were entered.

4. Execute the query and begin retrieving the results:

```
if ($r = mysqli_query($dbc, $query))
{
    while ($row = mysqli_fetch_
        array($r)) {
```

The **while** loop code was explained in Chapter 12, even though it wasn't used in that chapter. This construct is how you fetch every record returned by a query.

5. Begin printing the record:

```
print "<div><blockquote>
→ {$row['quote']}</blockquote>-
→ {$row['source']}\n";
```

This code starts a DIV, places the quotation itself within **blockquote** tags, and then shows the quote's attribution.

6. Indicate that the quotation is a favorite, if applicable:

```
if ($row['favorite'] == 1) {
    print ' <strong>Favorite!
→ </strong>';
}
```

The value of `$row['favorite']` will be either 1 or 0. If it's 1, the word *Favorite!*, emphasized, is displayed along with the record.

7. Add administrative links for editing and deleting the quote:

```
print "<p><b>Quote Admin:</b>
→ <a href=\"edit_quote.php?id=
→ {$row['quote_id']}\">Edit</a> <->
<a href=\"delete_quote.php?id=
→ {$row['quote_id']}\">Delete</a>
→ </p></div>\n";
```

For each quote, two links must be created. The first is to **edit\_quote.php** and the second to **delete\_quote.php**. Each link must also pass the **quote\_id** value along in the URL, as the code in Chapter 12 does **B**.

The end of the **print** statement closes the DIV for the specific quotation (begun in Step 5).

8. Complete the **while** loop and the **mysqli\_query()** conditional:

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

9. Close the database connection:

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

10. Complete the page:

```
include('includes/footer.php'); ?>
```

11. Save the view as **view\_quotes.php** and test in your Web browser.

**TIP** As some of the queries in this chapter demonstrate, you can use a column or value in an **ORDER BY** clause even if it's not selected by the query.

# Editing Quotes

The `view_quotes.php` page (and later, `index.php`) has links to `edit_quote.php`, where the administrator can update a quote. Functionally, this script will be very similar to `edit_entry.php` from Chapter 12:

1. The script needs to receive an ID value in the URL.
2. Using the ID, the record is retrieved and used to populate a form **A**.
3. Upon form submission, the form data will be validated (even if only slightly).
4. If the form data passes validation, the record will be updated in the database.

For the most part, this script is just another application of what you've already seen. But one new thing you'll learn here is how to check or not check a form's checkbox element based on a preexisting value.

## To create `edit_quote.php`:

1. Begin a new PHP document in your text editor or IDE, to be named `edit_quote.php` (Script 13.9):  

```
<?php // Script 13.9 -  
→ edit_quote.php  
define('TITLE', 'Edit a Quote');  
include('includes/header.php'); print  
'<h2>Edit a Quotation</h2>';
```
2. Terminate the script if the user isn't an administrator:  

```
if (!is_administrator()) {  
    print '<h2>Access Denied!</h2>  
→ <p class="error">You do not  
→ have permission to access  
→ this page.</p>';  
    include('includes/footer.php');  
    exit();  
}
```

## My Site of Quotes

### Edit a Quotation

It is the mark of an educated mind to be able to entertain a thought without accepting it.

Quote

Source

Is this a favorite? ☒

**A** The form's elements are prepopulated, and pre-checked, using the record's existing values.

**Script 13.9** The `edit_quote.php` script gives the administrator a way to update an existing record.

```
1  <?php // Script 13.9 - edit_quote.php  
2  /* This script edits a quote. */  
3  
4  / Define a page title and include the  
   header:  
5  define('TITLE', 'Edit a Quote');  
6  include('includes/header.php');  
7  
8  print '<h2>Edit a Quotation</h2>';  
9  
10 // Restrict access to administrators only:  
11 if (!is_administrator()) {  
12     print '<h2>Access Denied!</h2>  
    <p class="error">You do not have  
    permission to access this page.</  
    p>'; include('includes/footer.php');  
13     exit();  
14 }  
15  
16  
17 // Need the database connection:  
18 include('includes/mysql_connect.php');  
19  
20 if (isset($_GET['id']) && is_numeric  
    ($_GET['id']) && ($_GET['id'] > 0) ) { //  
    Display the entry in a form:  
21  
22     // Define the query.
```

*code continues on next page*



3. Include the database connection:

```
include('includes/mysql_connect.php');
```

Both phases of the script—displaying the form and handling the form—require a database connection, so the included file is incorporated at this point.

4. Validate that a numeric ID value was received in the URL:

```
if (isset($_GET['id']) &&
    → is_numeric($_GET['id']) &&
    → ($_GET['id'] > 0) ) {
```

This conditional is like one from Chapter 12, with the addition of checking that the ID value is greater than 0. Adding that clause doesn't do

anything for the security of the script—the `is_numeric()` test confirms that the value is safe to use in a query, but prevents the query from being executed if the ID has an unusable value.

5. Define and execute the query:

```
$query = "SELECT quote, source,
favorite FROM quotes WHERE
→ quote_id={$_GET['id']}";
if ($r = mysqli_query($dbc,$query
→ )) {
    $row = mysqli_fetch_array($r);
```

The query returns three columns for a specific record.

*continues on next page*

**Script 13.9** *continued*

```
23     $query = "SELECT quote, source, favorite FROM quotes WHERE quote_id={$_GET['id']}"; if
24     ($r = mysqli_query($dbc, $query)) { // Run the query.
25
26         $row = mysqli_fetch_array($r); // Retrieve the information.
27
28         // Make the form:
29         print '<form action="edit_quote.php" method="post">
30 <label>Quote <textarea name="quote" rows="5" cols="30">' . htmlentities($row['quote'])
31         . '</textarea></label></p>
32 <p><label>Source <input type="text" name="source" value="' .
33         htmlentities($row['source']) . '"></label></p>
34 <p><label>Is this a favorite? <input type="checkbox" name="favorite" value="yes">';
35
36         // Check the box if it is a favorite:
37         if ($row['favorite'] == 1) {
38             print ' checked="checked"';
39         }
40
41         // Complete the form:
42         print ' /></label></p>
43         <input type="hidden" name="id" value="' . $_GET['id'] . '">
44         <p><input type="submit" name="submit" value="Update This Quote!" /></p>
45 </form>';
46
47     } else { // Couldn't get the information.
48         print '<p class="error">Could not retrieve the quotation because:<br>' . mysqli_
49         error($dbc) . '</p><p>The query being run was: ' . $query . '</p>';
50     }
```

*code continues on next page*

6. Begin creating the form:

```
print '<form action="edit_quote.php"
→ method="post">
<p><label>Quote <textarea name=
→ "quote" rows="5" cols="30">' .
→ htmlentities($row['quote']) .
→ '</textarea></label></p>
```

```
<p><label>Source <input type= →
"text" name="source" value="" .
→ htmlentities($row['source']) .
→ "></label></p>
<p><label>Is this a favorite? →
<input type="checkbox"
→ name="favorite" value="yes">';
```

**Script 13.9** *continued*

```
49 } elseif (isset($_POST['id']) && is_numeric($_POST['id']) && ($_POST['id'] > 0)) { // Handle the form.
50
51     // Validate and secure the form data:
52     $problem = FALSE;
53     if ( !empty($_POST['quote']) && !empty($_POST['source']) ) {
54
55         // Prepare the values for storing:
56         $quote = mysql_real_escape_string(trim(strip_tags($_POST['quote'])), $dbc);
57         $source = mysql_real_escape_string(trim(strip_tags($_POST['source'])), $dbc);
58
59         // Create the "favorite" value:
60         if (isset($_POST['favorite'])) {
61             $favorite = 1;
62         } else {
63             $favorite = 0;
64         }
65
66     } else {
67         print '<p class="error">Please submit both a quotation and a source.</p>';
68         $problem = TRUE;
69     }
70
71     if (!$problem) {
72
73         // Define the query.
74         $query = "UPDATE quotes SET quote='$quote', source='$source', favorite=$favorite WHERE
quote_id={$_POST['id']}";
75         if ($r = mysql_query($query, $dbc)) {
76             print '<p>The quotation has been updated.</p>';
77         } else {
78             print '<p class="error">Could not update the quotation because:<br />' . mysql_
error($dbc) . '</p><p>The query being run was: ' . $query . '</p>';
79         }
80
81     } // No problem!
82
83 } else { // No ID set.
84     print '<p class="error">This page has been accessed in error.</p>';
85 } // End of main IF.
86
87 mysql_close($dbc); // Close the connection.
88
89 include('templates/footer.html'); // Include the footer.
90 ?>
```

The form is posted back to this same page. It starts with a text area, whose value will be prepopulated with the quote value retrieved from the database. That value is run through **htmlspecialchars()** as a safety precaution.

Next, a text input is created, prepopulated with the quotation's source. Finally, the checkbox for the indication of the favorite is begun. Note that this checkbox element is not completed, as the script needs to next determine whether or not to check the box (in Step 7).

7. Check the box if it is a favorite:

```
if ($row['favorite'] == 1) {  
    print ' checked="checked";'  
}
```

If the record's **favorite** value equals 1, then the administrator previously marked this quotation as a favorite. In that case, additional HTML needs to be added to the checkbox input to pre-check it. After this point in the code, the favorite checkbox's underlying HTML will be either

```
<input type="checkbox"  
→ name="favorite" value="yes"
```

or **B**

```
<input type="checkbox"  
→ name="favorite" value="yes"  
→ checked="checked"
```

*continues on next page*

```
<textarea name="quote" rows="5" cols="30">It is the mark of an educated mind to be ab.  
    <input type="text" name="source" value="Aristotle" /></label></p>  
s a favorite? <input type="checkbox" name="favorite" value="yes" checked="checked" />  
den" name="id" value="9" />
```

**B** The HTML source code for the page, upon first arriving, shows how the favorite checkbox can be pre-checked.

**8.** Complete the form:

```
print ' /></label></p>
<input type="hidden" name="id"
→ value="" . $_GET['id'] . "' />
<p><input type="submit" name=
→ "submit" value="Update This
→ Quote!"></p>
</form>;
```

The **print** statement starts by closing the checkbox. Then the form must also store the ID value in a hidden input, so that it's available to the script upon the form submission.

**9.** Create an error if the record could not be retrieved:

```
} else { // Couldn't get the
→ information.
    print '<p class="error">Could
→ not retrieve the quotation
→ because:<br>' . mysqli_
→ error($dbc) . ' .</p><p>The
→ query being run was: ' .
→ $query . '</p>';
}
```

**10.** Check for a form submission:

```
} elseif (isset($_POST['id']) &&
→ is_numeric($_POST['id']) &&
→ ($_POST['id'] > 0)) {
```

This conditional begins the second phase of the script: handling the submission of the form. The validation is the same as in Step 4, but now `$_POST['id']` is referenced instead of `$_GET['id']`.

**11.** Validate the form data:

```
$problem = FALSE;
if ( !empty($_POST['quote']) &&
→ !empty($_POST['source']) ) {
```

The form validation for the edit page mirrors that in the `add_quote.php` script (Script 13.7).

**12.** Prepare the values for use in the query:

```
$quote = mysqli_real_escape_string →
($dbc, trim(strip_tags($_POST
→ ['quote'])));
$source = mysqli_real_escape_
→ string($dbc, trim(strip_tags($_POST
→ ['source'])));
if (isset($_POST['favorite'])) {
    $favorite = 1;
} else {
    $favorite = 0;
}
```

This code is also taken straight from `add_quote.php`.

**13.** Indicate a problem if the form wasn't completed:

```
} else {
    print '<p class="error">Please
→ submit both a quotation and
→ a source.</p>';
    $problem = TRUE;
}
```

**14.** If no problem occurred, update the database:

```
if (!$problem) {
    $query = "UPDATE quotes SET
→ quote='$quote', source=
→ '$source', favorite=$favorite →
WHERE quote_id={$_POST['id']}";
    if ($r = mysqli_query($dbc, $query))
    {
        print '<p>The quotation has
→ been updated.</p>';
    }
```



❶ The result upon successfully editing a record.

The **UPDATE** query updates the values of three of the record's columns. The two string values are enclosed in single quotation marks (within the query); the numeric **\$favorite** value is not.

The **WHERE** clause, which dictates the record to be updated, is the critical piece.

Finally, a simple message indicates the success of the operation ❶.

15. Indicate a problem if the query failed:

```

} else {
    print '<p class="error">Could
    → not update the quotation
    → because:<br>' . mysqli_
    → error($dbc) . '</p><p>The
    → query being run was: ' .
    → $query . '</p>';
}

```

16. Complete the conditionals:

```

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

```

The **else** clause applies if no valid ID value is received by the page via either GET or POST.

17. Close the database connection and complete the page:

```

mysqli_close($dbc);
include('includes/footer.php'); ?>

```

18. Save the file and test in your Web browser (by clicking a link on **view\_quotes.php**).

# Deleting Quotes

The script for deleting existing quotations mimics `delete_entry.php` from Chapter 12. Upon first arriving, assuming that a valid record ID was passed along in the URL, the quote to be deleted is displayed **A**. If the administrator clicks the submit button, the form will be submitted back to this same page, at which point the record will be removed from the database **B**.

## To create `delete_quote.php`:

1. Begin a new PHP document in your text editor or IDE, to be named `delete_quote.php` (Script 13.10):  

```
<?php // Script 13.10 -  
→ delete_quote.php  
define('TITLE', 'Delete a Quote');  
include('includes/header.php');  
print '<h2>Delete a Quotation</h2>';
```

**Script 13.10** The `delete_quote.php` script provides the administrator with a way to delete an existing record.

```
1  <?php // Script 13.10 - delete_quote.php  
2  /* This script deletes a quote. */  
3  
4  / Define a page title and include the header:  
5  define('TITLE', 'Delete a Quote');  
6  include('includes/header.php');  
7  
8  print '<h2>Delete a Quotation</h2>';  
9  
10 // Restrict access to administrators only:  
11 if (!is_administrator()) {  
12     print '<h2>Access Denied!</h2>  
    <p class="error">You do not have permission to access this page.</p>';  
13     include('includes/footer.php');  
14     exit();  
15 }  
16  
17 // Need the database connection:18  
include('includes/mysql_connect.php');  
19  
20 if (isset($_GET['id']) && is_numeric($_GET['id']) && ($_GET['id'] > 0) ) { // Display the quote  
    in a form:
```

*code continues on next page*

## My Site of Quotes

### Delete a Quotation

Are you sure you want to delete this quote?

*It is the mark of an educated mind to be able to entertain a thought without accepting it.*

- Aristotle

Favorite!

Delete this Quote!

- A** The first step for deleting a record is confirming the record to be removed.

### Delete a Quotation

The quote entry has been deleted.

- B** Upon submission of the form, the quotation is deleted and a message is printed.

2. Terminate the script if the user isn't an administrator:

```
if (!is_administrator()) {  
    print '<h2>Access Denied!</h2>  
    → <p class="error">You do not  
    → have permission to access  
    → this page.</p>';  
    include('includes/footer.php');  
    exit();  
}
```

3. Include the database connection:

```
include('includes/mysql_connect.php');
```

*continues on next page*

#### Script 13.10 *continued*

```
21 // Define the query:  
22 $query = "SELECT quote, source, favorite FROM quotes WHERE  
23 quote_id={$GET['id']}";  
24 if ($r = mysqli_query($dbc, $query)) { // Run the query.  
25  
26     $row = mysqli_fetch_array($r); // Retrieve the information.  
27  
28     // Make the form:  
29     print '<form action="delete_quote.php" method="post">  
30     <p>Are you sure you want to delete this quote?</p>  
31     <div><blockquote> ' . $row['quote'] . '</blockquote>- ' . $row['source'];  
32  
33     // Is this a favorite?  
34     if ($row['favorite'] == 1) {  
35         print ' <strong>Favorite!</strong>';  
36     }  
37  
38     print '</div><br><input type="hidden" name="id" value="' . $GET['id'] . '">  
39 <p><input type="submit" name="submit" value="Delete this Quote!"></p>  
40 </form>';  
41  
42     } else { // Couldn't get the information.  
43         print '<p class="error">Could not retrieve the quote because:<br>' .  
44         mysqli_error($dbc) . '</p><p>The query being run was: ' . $query . '</p>';  
45     }  
46 } elseif (isset($_POST['id']) && is_numeric($_POST['id']) && ($_POST['id'] > 0) ) { // Handle the  
    form.  
47  
48     // Define the query:  
49     $query = "DELETE FROM quotes WHERE quote_id={$_POST['id']} LIMIT 1";
```

*code continues on next page*

4. Validate that a numeric ID value was received in the URL:

```
if (isset($_GET['id']) &&
    → is_numeric($_GET['id']) &&
    → ($_GET['id'] > 0) ) {
```

This is the same conditional used in `edit_quote.php`.

5. Retrieve the record to be deleted:

```
$query = "SELECT quote, source,
→ favorite FROM quotes WHERE
→ quote_id={$_GET['id']}";
if ($r = mysqli_query($dbc, $query)) →
{ $row = mysqli_fetch_array($r);
```

The standard three fields are retrieved from the database for the record.

Because only one record is being addressed in this script, the `mysqli_fetch_array()` function is called once, outside of any loop.

6. Begin creating the form:

```
print '<form action="delete_quote.
→ php" method="post">
<p>Are you sure you want to
→ delete this quote?</p>
<div><blockquote>' . $row['quote'] .
→ '</blockquote>- ' .
$row['source'];
```

The form, for the most part, just displays the quotation.

7. Indicate if the quotation is a favorite:

```
if ($row['favorite'] == 1) {
    print ' <strong>Favorite!
    → </strong>';
}
```

This code is the same as that on the `view_quotes.php` page, indicating that the quote is, in fact, a favorite.

#### Script 13.10 *continued*

```
50     $r = mysql_query($query, $dbc); // Execute the query.
51
52     // Report on the result:
53     if (mysqli_affected_rows($dbc) == 1)
54     { print '<p>The quote entry has been deleted.</p>';
55       } else {
56         print '<p class="error">Could not delete the blog entry because:<br />' .
           mysqli_error($dbc) . '</p><p>The query being run was: ' . $query . '</p>';
57       }
58
59     } else { // No ID received.
60       print '<p class="error">This page has been accessed in error.</p>';
61     } // End of main IF.
62
63     mysqli_close($dbc); // Close the connection.
64
65     include('includes/footer.php');
66     ?>
```



8. Complete the form:

```
print '</div><br />
<input type="hidden" name="id"
→ value="" . $_GET['id'] . "' />
<p><input type="submit" name=
→ "submit" value="Delete this
→ Quote!"></p>
</form>;
```

The form must contain a hidden input that will pass the quote ID back to the page upon form submission.

9. Complete the `mysqli_query()` conditional:

```
} else { // Couldn't get the
→ information.
    print '<p class="error">Could
→ not retrieve the quote
→ because:<br>' . mysqli_
→ error($dbc) . '</p><p>The
→ query being run was: ' .
→ $query . '</p>;'
}
```

If the query failed, the MySQL error, and the query itself, are displayed for debugging purposes.

10. Check for a form submission:

```
} elseif (isset($_POST['id']) &&
→ is_numeric($_POST['id']) &&
→ ($_POST['id'] > 0) ) {
```

This conditional begins the second phase of the script: handling the submission of the form. The validation is the same as in Step 4, but now `$_POST['id']` is referenced instead of `$_GET['id']`.

11. Delete the record:

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

The **DELETE** query will remove the record. The **WHERE** conditional indicates which specific record is to be removed, and the **LIMIT 1** clause is applied as an extra precaution.

12. Report on the result:

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

If the query succeeded, then one record will have been affected, and a message is displayed to the user **B**.

13. Complete the conditionals:

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

The **else** clause applies if no valid ID value is received by the page via either GET or POST.

14. Close the database connection and complete the page:

```
mysqli_close($dbc);
include('includes/footer.php'); ?>
```

15. Save the file and test in your Web browser (by clicking a link on **view\_quotes.php**).

# Creating the Home Page

Last, but certainly not least, there's the home page. For this site, the home page will be the only page used by the public at large. The home page will show a single quotation, but the specific quotation can be one of the following:

- The most recent (the default)
- A random quotation
- A random favorite quotation

To achieve this effect, links will pass different values in the URL back to this same page **A**.

The script should also display administrative links—edit and delete—for the currently displayed quote, if the user is an administrator **B**.

## To create index.php:

1. Begin a new PHP document in your text editor or IDE, to be named **index.php** (Script 13.11):

```
<?php // Script 13.11 - index.php
```

2. Include the header:

```
include('includes/header.php');
```

The home page does not need a custom title, so no constant is defined before including the header.

3. Include the database connection:

```
include('includes/mysql_connect.php');
```

4. Begin defining the query to be run:

```
if (isset($_GET['random'])) {  
    $query = 'SELECT quote_id,  
        → quote, source, favorite FROM  
        → quotes ORDER BY RAND() DESC  
        → LIMIT 1';
```



**A** Values passed in the URL trigger the execution of different queries.



**B** When an administrator views the home page, extra links are displayed.

**Script 13.11** The home page of the site shows a single quotation at a time, plus administrative links (when appropriate).

```
1 <?php // Script 13.11 - index.php  
2 /* This is the home page for this site.  
   It displays:  
3 - The most recent quote (default)  
4 - OR, a random quote  
5 - OR, a random favorite quote */  
6  
7 // Include the header:  
8 include('includes/header.php');  
9  
10 // Need the database connection: 11  
11 include('includes/mysql_connect.php');  
12  
13 // Define the query...
```

*code continues on next page*

If a `$_GET['random']` variable is set, the user clicked a link requesting a random quotation. It doesn't matter what value this variable has, so long as it is set.

For all of the queries, four columns—**quote\_id**, the **quote**, the **source**, and **favorite**—from one row will be returned. To only retrieve one row, a **LIMIT 1** clause is used.

To select a random row in MySQL, use the **ORDER BY RAND()** clause. This code uses MySQL's **RAND()** function, short for *random*, to return the records in a random order. So this query first selects every record in random order, and then returns only the first in that set.

5. Define the query that selects a random favorite record:

```
} elseif (isset($_GET['favorite'])) {  
    $query = 'SELECT quote_id,  
        → quote, source, favorite FROM  
        → quotes WHERE favorite=1 ORDER  
        → BY RAND() DESC LIMIT 1';
```

This query is similar to that in Step 4, but it uses a **WHERE** clause to restrict the pool of possible quotations to just those whose **favorite** value equals 1.

6. Define the default query:

```
} else {  
    $query = 'SELECT quote_id,  
        → quote, source, favorite FROM  
        → quotes ORDER BY date_entered  
        → DESC LIMIT 1';  
}
```

If no value was passed in the URL, then the home page should display the most recently added quotation. To do that, the query orders all of the quotes in descending order of date entered, and then limits the results to just a single record.

7. Execute the query and fetch the returned record:

```
if ($r = mysqli_query($dbc,$query)) {  
    $row = mysqli_fetch_array($r);
```

*continues on next page*

#### Script 13.11 *continued*

```
14 // Change the particulars depending upon values passed in the URL:  
15 if (isset($_GET['random'])) {  
16     $query = 'SELECT quote_id, quote, source, favorite FROM quotes ORDER BY RAND() DESC LIMIT 1';  
17 } elseif (isset($_GET['favorite'])) {  
18     $query = 'SELECT quote_id, quote, source, favorite FROM quotes WHERE favorite=1 ORDER BY  
        RAND() DESC LIMIT 1';  
19 } else {  
20     $query = 'SELECT quote_id, quote, source, favorite FROM quotes ORDER BY date_entered DESC  
        LIMIT 1';  
21 }  
22  
23 // Run the query:  
24 if ($r = mysqli_query($dbc, $query)) {  
25  
26     // Retrieve the returned record:  
27     $row = mysqli_fetch_array($r);  
28  
29     // Print the record:  
30     print "<div><blockquote>{$row['quote']}</blockquote>- {$row['source']}";
```

*code continues on next page*

8. Print the quotation:

```
print "<div><blockquote>{$row
→ ['quote']}</blockquote>-
→ {$row['source']}" ;
```

This code is similar to that in **view\_quotes.php**, but only needs to be used once.

9. Indicate if the quotation is a favorite and complete the DIV:

```
if ($row['favorite'] == 1) {
    print ' <strong>Favorite!
→ </strong>;
}
print '</div>;'
```

The conditional is the same as in **delete\_quote.php** and **view\_quotes.php**.

10. If the user is an administrator, create links to edit or delete this record:

```
if (is_administrator()) {
    print "<p><b>Quote Admin:</b>
→ <a href=\"edit_quote.php?
→ id={$row['quote_id']}\">Edit
→ </a> - + | + -
<a href=\"delete_quote.php?
→ id={$row['quote_id']}\">
→ Delete</a>
</p>\n";
}
```

If the user is an administrator, links to the edit and delete scripts will be added to the page. The links themselves have values just like those in **view\_quotes.php**.

**Script 13.11** *continued*

```
31
32     // Is this a favorite?
33     if ($row['favorite'] == 1) {
34         print ' <strong>Favorite!</strong>;
35     }
36
37     // Complete the DIV:
38     print '</div>;'
39
40     // If the admin is logged in, display admin links for this record:
41     if (is_administrator()) {
42         print "<p><b>Quote Admin:</b> <a href=\"edit_quote.php?id={$row['quote_id']}\">Edit</a> <->
43         <a href=\"delete_quote.php?id={$row['quote_id']}\">Delete</a>
44         </p>\n";
45     }
46
47 } else { // Query didn't run.
48     print '<p class="error">Could not retrieve the data because:<br />' . mysql_error($dbc) .
49     '</p><p>The query being run was: ' . $query . '</p>;'
50 } // End of query IF.
51
52 mysql_close($dbc); // Close the connection.
53
54 print '<p><a href="index.php">Latest</a> <-> <a href="index.php?random=true">Random</a> <->
55 <a href="index.php?favorite=true">Favorite</a><p>;'
56
57 include('templates/footer.html'); // Include the footer.
58 ?>
```

11. If the query didn't run, print an error message:

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

This code is for your own debugging purposes. You would not display a MySQL error, or the query that caused it, to the general public.

12. Close the database connection:

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

The database connection will no longer be needed, so it can be closed at this point.


13. Create links to other pages:

```
print '<p><a href="index.php">  
    → Latest</a> <-> <a href=  
    → "index.php?random=true">  
    → Random</a> <-> <a href=  
    → "index.php?favorite=true">  
    → Favorite</a><p>';
```

Three public links are added to the page, each back to this same script. The first link, which passes no values in the URL, will always show the most recent quotation. The second, which passes a *random* value in the URL, will trigger the query in Step 4, thereby retrieving a random record. The third link, which passes a *favorite* value in the URL, will trigger the query in Step 5, thereby retrieving a random favorite record.


14. Include the footer and complete the page:

```
include('templates/footer.html');  
?>
```

15. Save the file and test in your Web browser .

**TIP** Normally the home page is one of the first scripts written, not the last. But in this case I wanted to build up to this point in the example.



-  The latest quotation (note the URL).