KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Module 4

Syllabus

- Advanced PHP: Form processing and Business Logic-Cookies- Sessions & MySQL Integration-Connecting to MySQL with PHP- Performing CREATE, DELETE, INSERT, SELECT and UPDATE operations on MySQL table -Working with MySQL data-Reading from Database -Dynamic Content.

Form processing and Business Logic

19.8.1 Superglobal Arrays

Knowledge of a client's execution environment is useful to system administrators who want to access client-specific information such as the client's web browser, the server name or the data sent to the server by the client. One way to obtain this data is by using a **superglobal array**. Superglobal arrays are associative arrays predefined by PHP that hold variables acquired from user input, the environment or the web server, and are accessible in any variable scope. Some of PHP's superglobal arrays are listed in Fig. 19.12.

Variable name	Description
`$_SERVER`	Data about the currently running server.
`$_ENV`	Data about the client's environment.
`$_GET`	Data sent to the server by a `get` request.
`$_POST`	Data sent to the server by a `post` request.
`$_COOKIE`	Data contained in cookies on the client's computer.
`$GLOBALS`	Array containing all global variables.

Fig. 19.12 | Some useful superglobal arrays.

Superglobal arrays are useful for verifying user input. The arrays `$_GET` and `$_POST` retrieve information sent to the server by HTTP `get` and `post` requests, respectively, making it possible for a script to have access to this data when it loads another page. For example, if data entered by a user into a form is posted to a script, the `$_POST` array will contain all of this information in the new script. Thus, any information entered into the form can be accessed easily from a confirmation page, or a page that verifies whether fields have been entered correctly.

Form processing and Business Logic

19.8.2 Using PHP to Process HTML5 Forms

Forms enable web pages to collect data from users and send it to a web server for processing. Such capabilities allow users to purchase products, request information, send and receive web-based e-mail, create profiles in online networking services and take advantage of various other online services. The HTML5 form in Fig. 19.13 gathers information to add a user to a mailing list.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.13: form.html -->
4  <!-- HTML form for gathering user input. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Sample Form</title>
9        <style type = "text/css">
10          label  { width: 5em; float: left; }
11       </style>
12    </head>
13    <body>
14       <h1>Registration Form</h1>
15       <p>Please fill in all fields and click Register.</p>
16
17       <!-- post form data to form.php -->
18       <form method = "post" action = "form.php">
19          <h2>User Information</h2>
20
21          <!-- create four text boxes for user input -->
22          <div><label>First name:</label>
23             <input type = "text" name = "fname"></div>
24          <div><label>Last name:</label>
25             <input type = "text" name = "lname"></div>
26          <div><label>Email:</label>
27             <input type = "text" name = "email"></div>
28          <div><label>Phone:</label>
29             <input type = "text" name = "phone"
30                placeholder = "(555) 555-5555"></div>
31          </div>
32
33          <h2>Publications</h2>
34          <p>Which book would you like information about?</p>
35
36          <!-- create drop-down list containing book names -->
37          <select name = "book">
38             <option>Internet and WWW How to Program</option>
```

Fig. 19.13 | HTML5 form for gathering user input (Part 1 of 2.)

Form processing and Business Logic

```
39             <option>C++ How to Program</option>
40             <option>Java How to Program</option>
41             <option>Visual Basic How to Program</option>
42          </select>
43
44          <h2>Operating System</h2>
45          <p>Which operating system do you use?</p>
46
47          <!-- create five radio buttons -->
48          <p><input type = "radio" name = "os" value = "Windows"
49             checked>Windows
50             <input type = "radio" name = "os" value = "Mac OS X">Mac OS X
51             <input type = "radio" name = "os" value = "Linux">Linux
52             <input type = "radio" name = "os" value = "Other">Other</p>
53
54          <!-- create a submit button -->
55          <p><input type = "submit" name = "submit" value = "Register"></p>
56       </form>
57    </body>
58 </html>
```

The form is filled out with an incorrect phone number

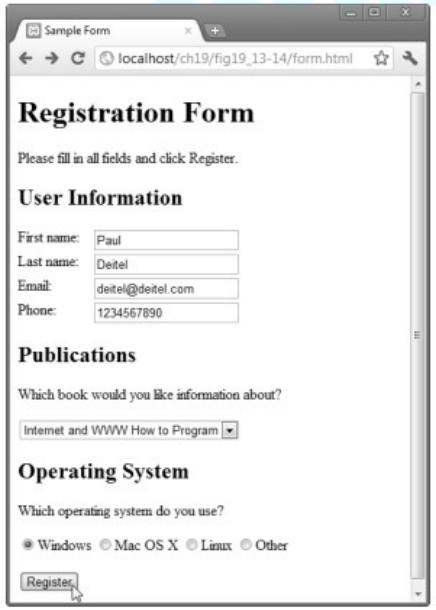


Fig. 19.13 | HTML5 form for gathering user inpu

Form processing and Business Logic

The `form`'s `action` attribute (line 18) indicates that when the user clicks the **Register** button, the `form` data will be `posted` to `form.php` (Fig. 19.14) for processing. Using `method = "post"` appends form data to the browser request that contains the protocol (i.e., HTTP) and the URL of the requested resource (specified by the `action` attribute). Scripts located on the web server's machine can access the form data sent as part of the request.

We assign a unique `name` (e.g., `email`) to each of the `form`'s controls. When **Register** is clicked, each field's `name` and `value` are sent to the web server. Script `form.php` accesses the value for each field through the superglobal array `$_POST`, which contains key/value pairs corresponding to name–value pairs for variables submitted through the form. [*Note:* The superglobal array `$_GET` would contain these key–value pairs if the form had been submitted using the HTTP *get* method. In general, `get` is not as secure as `post`, because it appends the information directly to the URL, which is visible to the user.] Figure 19.14 processes the data `posted` by `form.html` and sends HTML5 back to the client.

Form processing and Business Logic

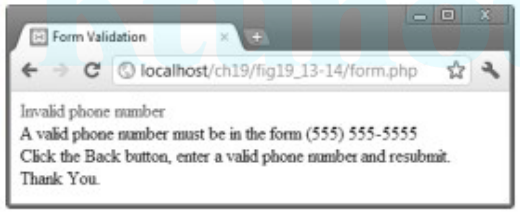
```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.14: form.php -->
4  <!-- Process information sent from form.html. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Form Validation</title>
9        <style type = "text/css">
10          p       { margin: 0px; }
11          .error  { color: red }
12          p.head  { font-weight: bold; margin-top: 10px; }
13       </style>
14    </head>
15    <body>
16       <?php
17          // determine whether phone number is valid and print
18          // an error message if not
19          if (!preg_match( "/^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$/",
20             $_POST["phone"]))
21          {
22             print( "<p class = 'error'>Invalid phone number</p>
23                <p>A valid phone number must be in the form
24                (555) 555-5555</p><p>Click the Back button,
25                enter a valid phone number and resubmit.</p>
26                <p>Thank You.</p></body></html>" );
27             die(); // terminate script execution
28          }
29       ?><!-- end PHP script -->
30       <p>Hi <?php print( $_POST["fname"] ); ?>. Thank you for
31          completing the survey. You have been added to the
```

Fig. 19.14 | Process information sent from `form.html`. (Part 1 of 2.)

Form processing and Business Logic

```
32          <?php print( $_POST["book"] ); ?>mailing list.</p>
33       <p class = "head">The following information has been saved
34          in our database:</p>
35       <p>Name: <?php print( $_POST["fname"] );
36          print( $_POST["lname"] ); ?></p>
37       <p>Email: <?php print( "$email" ); ?></p>
38       <p>Phone: <?php print( "$phone" ); ?></p>
39       <p>OS: <?php print( $_POST["os"] ); ?></p>
40       <p class = "head">This is only a sample form.
41          You have not been added to a mailing list.</p>
42    </body>
43 </html>
```

a) Submitting the form in Fig. 19.13 redirects the user to `form.php`, which gives appropriate instructions if the phone number is in an incorrect format



b) The results of form.php after the user submits the form in Fig. 19.13 with a phone number in a valid format

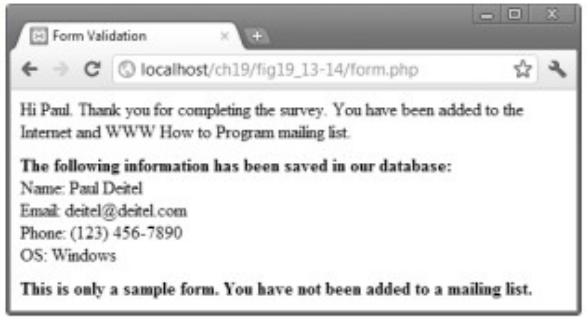


Fig. 19.14 | Process information sent from `form.html`. (Part 2 of 2.)

Form processing and Business Logic

Lines 19–20 determine whether the phone number entered by the user is valid. We get the phone number from the `$_POST` array using the expression `$_POST["phone"]`, where `"phone"` is the name of the corresponding `input` field in the form. The validation in this example requires the phone number to begin with an opening parenthesis, followed by an area code, a closing parenthesis, a space, an exchange, a hyphen and a line number. It's crucial to validate information that will be entered into databases or used in mailing lists. For example, validation can be used to ensure that credit card numbers contain the proper number of digits before the numbers are encrypted and sent to a merchant. This script implements the business logic, or business rules, of our application.

Form processing and Business Logic

In lines 19–20, the expression `\(` matches the opening parenthesis of the phone number. We want to match the literal character `(`, so we escape its normal meaning by preceding it with the backslash character (`\`). This parenthesis in the expression must be followed by three digits (`[0-9]{3}`), a closing parenthesis, three more digits, a literal hyphen and four additional digits. Note that we use the `^` and `$` symbols to ensure that no extra characters appear at either end of the string.

If the regular expression is matched, the phone number has a valid format, and an HTML5 document is sent to the client that thanks the user for completing the form. We extract each `input` element's value from the `$_POST` array in lines (30–39). Otherwise, the body of the `if` statement executes and displays an error message.

Function **`die`** (line 27) *terminates* script execution. This function is called if the user did not enter a correct telephone number, since we do not want to continue executing the rest of the script. The function's optional argument is a string or an integer. If it's a string, it's printed as the script exits. If it's an integer, it's used as a return status code (typically in command-line PHP shell scripts).

Form processing and Business Logic

19.9 Reading from a Database

PHP offers built-in support for many databases. Our database examples use MySQL. We assume that you've followed the XAMPP installation instructions in Chapter 17 (XAMPP includes MySQL) and that you've followed the Chapter 18 instructions for setting up a MySQL user account and for creating the databases we use in this chapter.

The example in this section uses a `Products` database. The user selects the name of a column in the database and submits the form. A PHP script then builds a SQL `SELECT` query, queries the database to obtain the column's data and outputs the data in an HTML5 document that's displayed in the user's web browser. Chapter 18 discusses how to build SQL queries.

Figure 19.15 is a web page that *posts form data* consisting of a selected database column to the server. The script in Fig. 19.16 *processes the form data*.

HTML5 Document

Line 12 of Fig. 19.15 begins an HTML5 `form`, specifying that the data submitted from the `form` will be sent to the script `database.php` (Fig. 19.16) in a `post` request. Lines 16–22 add a select box to the `form`, set the name of the select box to `select` and set its default selection to `*`. Submitting `*` specifies that *all* rows and columns are to be retrieved from the database. Each of the database's column names is set as an option in the select box.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.15: data.html -->
4  <!-- Form to query a MySQL database. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Sample Database Query</title>
9     </head>
```

ig. 19.15 | Form to query a MySQL database. (Part 1 of 2.)

Form processing and Business Logic

```
10      <body>
11         <h1>Querying a MySQL database.</h1>
12         <form method = "post" action = "database.php">
13            <p>Select a field to display:
14               <!-- add a select box containing options -->
15               <!-- for SELECT query -->
16               <select name = "select">
17                  <option selected>*</option>
18                  <option>ID</option>
19                  <option>Title</option>
20                  <option>Category</option>
21                  <option>ISBN</option>
22               </select></p>
23            <p><input type = "submit" value = "Send Query"></p>
24         </form>
25      </body>
26   </html>
```

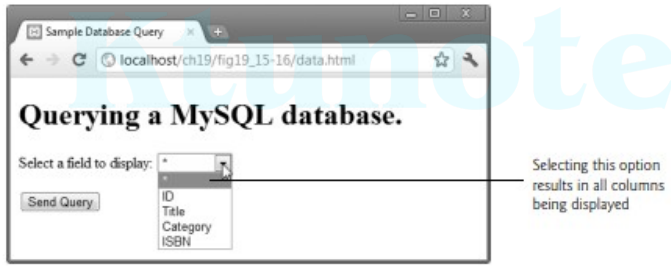


Fig. 19.15 | Form to query a MySQL database. (Part 2 of 2.)

database.php

Script `database.php` (Fig. 19.16) builds a SQL query with the `posted` field name then queries the MySQL database. Line 25 concatenates the `posted` field name to a `SELECT` query. Lines 28–29 call function **`mysql_connect`** to connect to the MySQL database. We pass three arguments—the server's hostname, a username and a password. The host name `localhost` is your computer. The username and password specified here were created in Chapter 18. Function `mysql_connect` returns a **database handle**—a representation of PHP's connection to the database—which we assign to variable `$database`. If the connection to MySQL fails, the function returns `false` and we call `die` to output an error message and terminate the script. Line 33 calls function **`mysql_select_db`** to select and open the database to be queried (in this case, `products`). The function returns `true` on success or `false` on failure. We call `die` if the database cannot be opened.

```
<!DOCTYPE html>

<!-- Fig. 19.16: database.php -->
```

Fig. 19.16 | Querying a database and displaying the results. (Part 1 of 3.)

```
<!-- Querying a database and displaying the results. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Search Results</title>
   <style type = "text/css">
         body  { font-family: sans-serif;
                 background-color: lightyellow; }
         table { background-color: lightblue;
                 border-collapse: collapse;
                 border: 1px solid gray; }
         td    { padding: 5px; }
         tr:nth-child(odd) {
                 background-color: white; }
      </style>
   </head>
   <body>
      <?php
         $select = $_POST["select"]; // creates variable $select

         // build SELECT query
         $query = "SELECT " . $select . " FROM books";

         // Connect to MySQL
         if ( !( $database = mysql_connect( "localhost",
            "iw3htp", "password" ) ) )
            die( "Could not connect to database </body></html>" );

         // open Products database
         if ( !mysql_select_db( "products", $database ) )
            die( "Could not open products database </body></html>" );

         // query Products database
         if ( !( $result = mysql_query( $query, $database ) ) )
         {
            print( "<p>Could not execute query!</p>" );
            die( mysql_error() . "</body></html>" );
         } // end if

         mysql_close( $database );
      ?><!-- end PHP script -->
      <table>
         <caption>Results of "SELECT <?php print( "$select" ) ?>
            FROM books"</caption>
         <?php
            // fetch each record in result set
            while ( $row = mysql_fetch_row( $result ) )
            {
               // build table to display results
               print( "<tr>" );

               foreach ( $row as $key => $value )
                  print( "<td>$value</td>" );
```

Fig. 19.16 | Querying a database and displaying the results. (Part 2 of 3.)

```
57
58                print( "</tr>" );
59             } // end while
60          ?><!-- end PHP script -->
61       </table>
62       <p>Your search yielded
63          <?php print( mysql_num_rows( $result ) ) ?> results.</p>
64       <p>Please email comments to <a href = "mailto:deitel@deitel.com">
65             Deitel and Associates, Inc.</a></p>
66    </body>
67 </html>
```

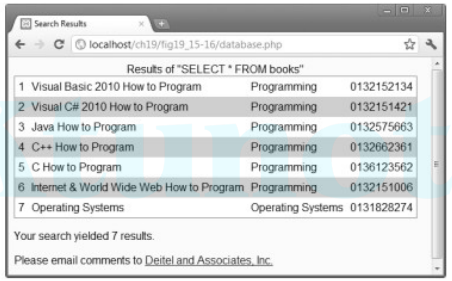


Fig. 19.16 | Querying a database and displaying the results. (Part 3 of 3.)

To query the database, line 37 calls function **`mysql_query`**, specifying the query string and the database to query. If the query fails, the function returns `false`. Function `die` is then called with a call to function **`mysql_error`** as an argument. Function `mysql_error` returns any error strings from the database. If the query succeeds, `mysql_query` returns a `resource` containing the query result, which we assign to variable `$result`. Once we've stored the data in `$result`, we call **`mysql_close`** in line 43 to close the connection to the database. Function `mysql_query` can also execute SQL statements such as `INSERT` or `DELETE` that do not return results.

Lines 50–59 iterate through each record in the *result set* and construct an HTML5 table containing the results. The loop's condition calls the **`mysql_fetch_row`** function to return an array containing the values for each column in the current row of the query result (`$result`). The array is stored in variable `$row`. Lines 55–56 construct individual cells for each column in the row. The `foreach` statement takes the name of the array (`$row`), iterates through each index value of the array and stores the value in variable `$value`. Each element of the array is then printed as an individual cell. When the result has no more rows, `false` is returned by function `mysql_fetch_row`, which terminates the loop.

After all the rows in the result have been displayed, the table's closing tag is written (line 61). Lines 62–63 display the number of rows in `$result` by calling **`mysql_num_rows`** with `$result` as an argument.

Cookies(Textbook 1-DEITAL)

A **cookie** is a piece of information that's stored by a server in a text file on a client's computer to maintain information about the client during and between browsing sessions. A website can store a cookie on a client's computer to record user preferences and other information that the website can retrieve during the client's subsequent visits. For example, a website can use cookies to store clients' zip codes, so that it can provide weather reports and news updates tailored to the user's region. Websites also can use cookies to track information about client activity. Analysis of information collected via cookies can reveal the popularity of websites or products. Marketers can use cookies to determine the effectiveness of advertising campaigns.

Websites store cookies on users' hard drives, which raises issues regarding security and privacy. Websites should not store critical information, such as credit card numbers or passwords, in cookies, because cookies are typically stored in text files that any program can read. Several cookie features address security and privacy concerns. *A server can access only the cookies that it has placed on the client.* For example, a web application running on `www.deitel.com` cannot access cookies that the website `www.pearson.com` has placed on the client's computer. A cookie also has an *expiration date*, after which the web browser deletes it. Users who are concerned about the privacy and security implications of cookies can disable cookies in their browsers. But, disabling cookies can make it difficult or impossible for the user to interact with websites that rely on cookies to function properly.

The information stored in a cookie is sent back to the web server from which it originated whenever the user requests a web page from that particular server. The web server can send the client HTML5 output that reflects the preferences or information that's stored in the cookie.

HTML5 Document

Figure 19.17 presents an HTML5 document containing a form in which the user specifies a name, height and favorite color. When the user clicks the **Write Cookie** button, the `cookies.php` script (Fig. 19.18) executes.

Writing Cookies: `cookies.php`

Script `cookies.php` (Fig. 19.18) calls function **`setcookie`** (lines 8–10) to set the cookies to the values `posted` from `cookies.html`. The cookies defined in function `setcookie` are sent to the client at the same time as the information in the HTTP header; therefore, `setcookie` needs to be called *before* any other output. Function `setcookie` takes the name of the cookie to be set as the first argument, followed by the value to be stored in the cookie. For example, line 8 sets the name of the cookie to `"Name"` and the value to variable `$Name`, which is passed to the script from `cookies.html`. The optional third argument indicates the expiration date of the cookie. In this example, we set the cookies to expire in five days by taking the current time, which is returned by function **`time`**, and adding the constant `FIVE_DAYS`—the number of seconds after which the cookie is to expire (`60` seconds per minute * `60` minutes per hour * `24` hours per day * `5` = `5` days). *If no expiration date is specified, the cookie lasts only until the end of the current session*—that is, when the user closes the browser. This type of cookie is known as a **session cookie**, while one with an expiration date is a **persistent cookie**. If only the `name` argument is passed to function `setcookie`, the cookie is deleted from the client's computer. Lines 13–35 send a web page to the client

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.17: cookies.html -->
4  <!-- Gathering data to be written as a cookie. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Writing a cookie to the client computer</title>
9        <style type = "text/css">
10          label { width: 7em; float: left; }
11       </style>
12    </head>
13    <body>
14       <h2>Click Write Cookie to save your cookie data.</h2>
15       <form method = "post" action = "cookies.php">
16          <div><label>Name:</label>
17             <input type = "text" name = "name"><div>
18          <div><label>Height:</label>
19             <input type = "text" name = "height"></div>
20          <div><label>Favorite Color:</label>
21             <input type = "text" name = "Color"></div>
22          <p><input type = "submit" value = "Write Cookie">
23       </form>
24    </body>
25 </html>
```

Fig. 19.17 | Gathering data to be written as a cookie.

indicating that the cookie has been written and listing the values that are stored in the cookie.

```php
<!-- Fig. 19.18: cookies.php -->
<!-- Writing a cookie to the client. -->
<?php
   define( "FIVE_DAYS", 60 * 60 * 24 * 5 ); // define constant

   // write each form field's value to a cookie and set the
   // cookie's expiration date
   setcookie( "name", $_POST["name"], time() + FIVE_DAYS );
   setcookie( "height", $_POST["height"], time() + FIVE_DAYS );
   setcookie( "color", $_POST["color"], time() + FIVE_DAYS );
?><!-- end PHP script -->

<!DOCTYPE html>

<html>
   <head>
      <meta charset = "utf-8">
      <title>Cookie Saved</title>
      <style type = "text/css">
         p { margin: 0px; }
      </style>
   </head>
   <body>
      <p>The cookie has been set with the following data:</p>

      <!-- print each form field's value -->
      <p>Name: <?php print( $Name ) ?></p>
      <p>Height: <?php print( $Height ) ?></p>
      <p>Favorite Color:
         <span style = "color: <?php print( "$Color" ) ?> ">
         <?php print( "$Color" ) ?></span></p>
      <p>Click <a href = "readCookies.php">here</a>
         to read the saved cookie.</p>
   </body>
</html>
```

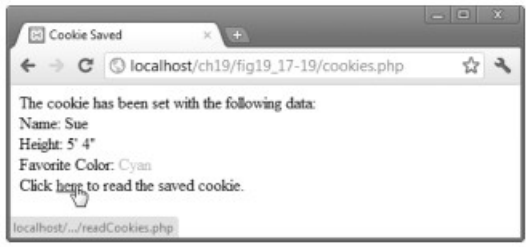


Fig. 19.18 | Writing a cookie to the client.

Reading an Existing Cookie

Figure 19.19 reads the cookie that was written in Fig. 19.18 and displays the cookie's information in a table. PHP creates the superglobal array **`$_COOKIE`**, which contains all the

cookie values indexed by their names, similar to the values stored in array `$_POST` when an HTML5 form is posted (see Section 19.8).

Lines 18–19 of Fig. 19.19 iterate through the `$_COOKIE` array using a `foreach` statement, printing out the name and value of each cookie in a paragraph. The `foreach` statement takes the name of the array (`$_COOKIE`) and iterates through each index value of the array (`$key`). In this case, the index values are the names of the cookies. Each element is then stored in variable `$value`, and these values become the individual cells of the table. Try closing your browser and revisiting `readCookies.php` to confirm that the cookie has persisted.

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.19: readCookies.php -->
4  <!-- Displaying the cookie's contents. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Read Cookies</title>
9        <style type = "text/css">
10          p { margin: 0px; }
11       </style>
12    </head>
13    <body>
14       <p>The following data is saved in a cookie on your computer.</p>
15       <?php
16          // iterate through array $_COOKIE and print
17          // name and value of each cookie
18          foreach ($_COOKIE as $key => $value )
19             print( "<p>$key: $value</p>" );
20       ?><!-- end PHP script -->
21    </body>
22 </html>
```

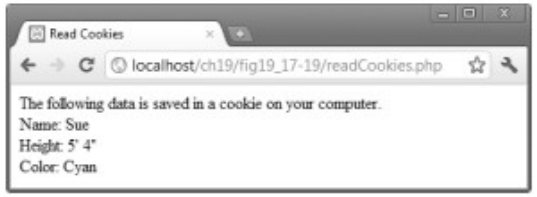


Fig. 19.19 | Displaying the cookie's contents.

Cookies

- You can use cookies within your PHP scripts to store small bits of information about a user.
- A cookie is a small amount of data stored by the user's browser in compliance with a request from a server or script.
- A single host can request that up to 20 cookies be stored by a user's browser.
- Each cookie consists of a name, value, and expiration date, as well as host and path information. The size of an individual cookie is limited to 4KB. After a cookie is set, only the originating host can read the data, ensuring that the user's privacy is respected. Furthermore, users can configure their browser to notify them upon receipt of all cookies, or even to refuse all cookie requests. For this reason, cookies should be used in moderation and should not be relied on as an essential element of an environment design without first warning users.

- A PHP script that sets a cookie might send headers that look something like this:

```
HTTP/1.1 200 OK
Date: Wed, 18 Jan 2012 10:50:58 GMT
Server: Apache/2.2.21 (Unix) PHP/5.4.0
X-Powered-By: PHP/5.4.0
Set-Cookie: vegetable=artichoke; path=/; domain=yourdomain.com
Connection: close
Content-Type: text/html
```

- As you can see, this Set-Cookie header contains a name/value pair, a path, and a domain. If set, the expiration field provides the date at which the browser should “forget” the value of the cookie. If no expiration date is set, the cookie expires when the user’s session expires—that is, when he closes his browser.

- The path and domain fields work together: The path is a directory found on the domain, below which the cookie should be sent back to the server. If the path is “/”, which is common, that means the cookie can be read by any files below the document root. If the path is “/products/”, the cookie can be read only by files within the /products directory of the website

- The domain field represents the Internet domain from which cookie-based communication is allowed. For example, if your domain is www.yourdomain.com and you use www.yourdomain.com as the domain value for the cookie, the cookie will be valid only when browsing the www.domain.com website. This could pose a problem if you send the user to some domain like www2.domain.com or billing.domain.com within the course of his browsing experience, because the original cookie will no longer work. Therefore, it is common simply to begin the value of the domain slot in cookie definitions with a dot, leaving off the host (for example, .domain.com). In this manner, the cookie is valid for all hosts on the domain. The domain cannot be different from the domain from which the cookie was sent; otherwise, the cookie will not function properly, if at all, or the web browser will refuse the cookie in its entirety

Accessing Cookies

- If your web browser is configured to store cookies, it keeps the cookie-based information until the expiration date. If the user points the browser at any page that matches the path and domain of the cookie, it resends the cookie to the server. The browser's headers might look something like this:


```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7 (KHTML, like
  Gecko) Chrome/16.0.912.75 Safari/535.7
Host: www.yourdomain.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en,pdf
Accept-Charset: iso-8859-1,*,utf-8
Cookie: vegetable=artichoke
```

A PHP script then has access to the cookie in the environment variable `HTTP_COOKIE` or as part of the `$_COOKIE` superglobal variable, which you may access three different ways:

```
echo  $_SERVER['HTTP_COOKIE'];   // will print "vegetable=artichoke"
echo  getenv('HTTP_COOKIE');  // will print "vegetable=artichoke"
echo  $ COOKIE['vegeta
```

Setting a Cookie with PHP

- You can set a cookie in a PHP script in two ways. First, you can use the header() function to set the Set-Cookie header. The header() function requires a string that is then included in the header section of the server response. Because headers are sent automatically for you, header() must be called before any output at all is sent to the browser:

```
header("Set-Cookie: vegetable=artichoke; expires=Thu, 19-Jan-12 14:39:58 GMT;
path=/; domain=yourdomain.com");
```

- The setcookie() function does what its name suggests—it outputs a Set-Cookie header. For this reason, it should be called before any other content is sent to the browser. The function accepts the cookie name, cookie value, expiration date in UNIX epoch format, path, domain, and integer that should be set to 1 if the cookie is to be sent only over a secure connection. All arguments to this function are optional apart from the first (cookie name) parameter.

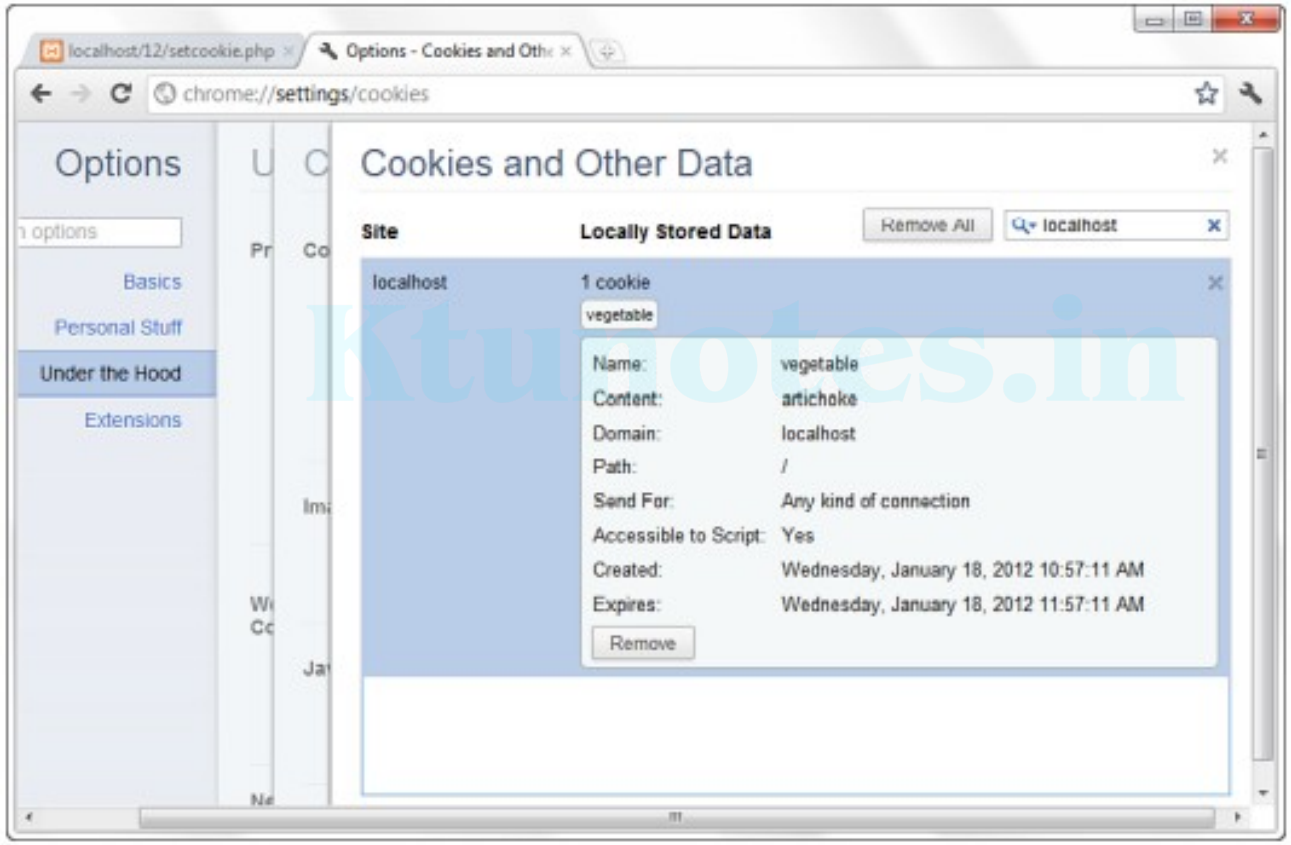
LISTING 12.1 Setting and Printing a Cookie Value

```
1: <?php
2: setcookie("vegetable", "artichoke", time()+3600, "/", ".yourdomain.com", 0);
3:
4: if (isset($_COOKIE['vegetable'])) {
5:   echo "<p>Hello again! You have chosen:  ".$_COOKIE['vegetable']."</p>";
6: } else {
7:   echo "<p>Hello, you. This may be your first visit.</p>";
8: }
9: ?>
```

- Even though the listing sets the cookie (line 2) when the script is run for the first time, the $_COOKIE[‘vegetable’] variable is not created at this point. Because a cookie is read only when the browser sends it to the server, you cannot read it until the user revisits a page within this domain

- The cookie name is set to “vegetable” on line 2, and the cookie value to “artichoke”.
- The time() function gets the current timestamp and adds 3600 to it (3,600 seconds in an hour). This total represents the expiration date. The code defines a path of “/”, which means that a cookie should be sent for any page within this server environment.
- The domain argument is set to “.yourdomain.com” (you should make the change relevant to your own domain or leave it blank if you are working on localhost), which means that a cookie will be sent to any server in that group.
- Finally, the code passes 0 to setcookie(), signaling that cookies can be sent in an unsecure environment. Passing setcookie() an empty string (“”) for string arguments or 0 for integer fields causes these arguments to be skipped

Viewing a stored cookie in a web browser.



Deleting a Cookie with PHP

Officially, to delete a cookie, you call `setcookie()` with the name argument only:

```
setcookie("vegetable");
```

This approach does not always work well, however, and you should not rely on it. Instead, to delete a cookie, it is safest to set the cookie with a date that you are sure has already expired:

```
setcookie("vegetable", "", time()-60, "/", ".yourdomain.com", 0);
```

Also make sure that you pass `setcookie()` the same path, domain, and secure parameters as you did when originally setting the cookie.

Session Function Overview

- Session functions provide a unique identifier to a user, which can then be used to store and acquire information linked to that ID.
- When a visitor accesses a session enabled page, either a new identifier is allocated or the user is reassociated with one that was already established in a previous visit.
- Any variables that have been associated with the session become available to your code through the $_SESSION superglobal.
- Session state is usually stored in a temporary file, although you can implement database storage or other server-side storage methods using a function called session_set_save_handler().

Starting a Session

To work with a session, you need to explicitly start or resume that session *unless* you have changed your `php.ini` configuration file. By default, sessions do not start automatically. If you want to start a session this way, you must find the following line in your `php.ini` file and change the value from `0` to `1` (and restart the web server):

```
session.auto_start = 0
```

By changing the value of `session.auto_start` to `1`, you ensure that a session initiates for every PHP document. If you don't change this setting, you need to call the `session_start()` function in each script.

After a session is started, you instantly have access to the user's session ID via the `session_id()` function. The `session_id()` function enables you to either set or retrieve a session ID. Listing 12.2 starts a session and prints the session ID to the browser.

LISTING 12.2 Starting or Resuming a Session

```
1: <?php
2: session_start();
3: echo "<p>Your session ID is ".session_id().".</p>";
4: ?>
```

When this script (let's call it `session_checkid.php`) is run for the first time from a browser, a session ID is generated by the `session_start()` function call on line 2. If the script is later reloaded or revisited, the same session ID is allocated to the user. This action assumes that the user has cookies enabled. For example, when I run this script the first time, the output is as follows:

```
Your session ID is 8jou17in51d08e5onsjkbles16.
```

When I reload the page, the output is still

```
Your session ID is 8jou17in51d08e5onsjkbles16.
```

because I have cookies enabled and the session ID still exists.

Because `start_session()` attempts to set a cookie when initiating a session for the first time, it is imperative that you call this function before you output anything else at all to the browser. If you do not follow this rule, your session will not be set, and you will likely see warnings on your page.

Sessions remain current as long as the web browser is active. When the user restarts the browser, the cookie is no longer stored. You can change this behavior by altering the `session.cookie_lifetime` setting in your `php.ini` file. The default value is `0`, but you can set an expiry period in seconds.

Working with Session Variables

Accessing a unique session identifier in each of your PHP documents is only the start of session functionality. When a session is started, you can store any number of variables in the `$_SESSION` superglobal and then access them on any session-enabled page.

Listing 12.3 adds two variables into the `$_SESSION` superglobal: `product1` and `product2` (lines 3 and 4).

LISTING 12.3 Storing Variables in a Session

```
1: <?php
2: session_start();
3: $_SESSION['product1'] = "Sonic Screwdriver";
4: $_SESSION['product2'] = "HAL 2000";
5: echo "The products have been registered.";
6: ?>
```

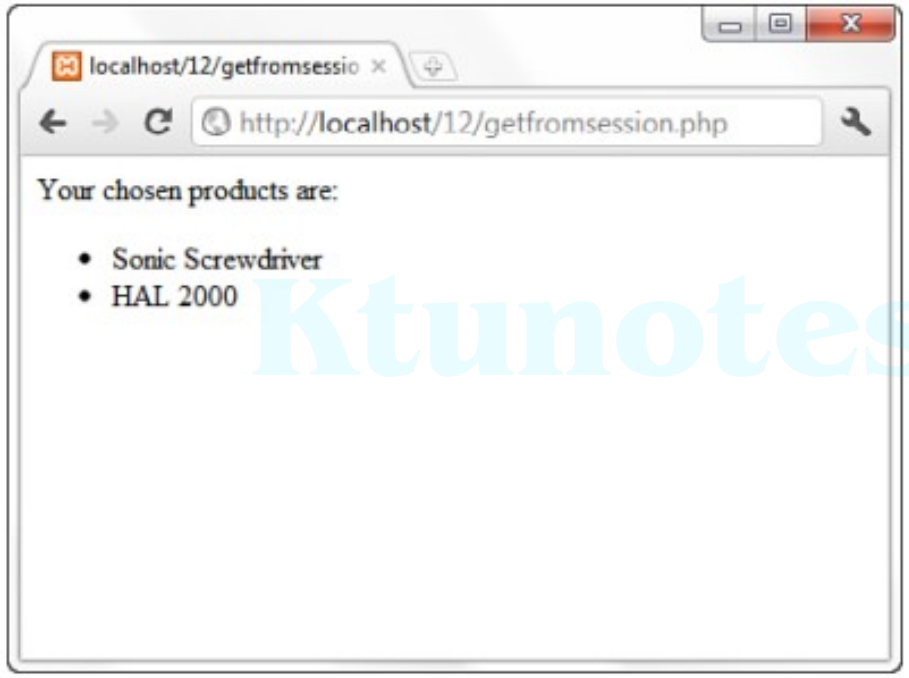
The magic in Listing 12.3 will not become apparent until the user moves to a new page. Listing 12.4 creates a separate PHP script that accesses the variables stored in the `$_SESSION` superglobal.

LISTING 12.4 Accessing Stored Session Variables

```
1: <?php
2: session_start();
3: ?>
4: <p>Your chosen products are:</p>
5: <ul>
6: <li><?php echo $_SESSION['product1']; ?></li>
7: <li><?php echo $_SESSION['product2']; ?></li>
8: </ul>
```

Figure 12.2 shows the output from Listing 12.4. As you can see, you have access to the `$_SESSION['product1']` and `$_SESSION['product2']` variables in an entirely new page.

Accessing the stored session variables



Although not a terribly interesting or useful example, the script does show how to access stored session variables. Behind the scenes, PHP writes information to a temporary file. You can find out where this file is being written on your system by using the `session_save_path()` function. This function optionally accepts a path to a directory and then writes all session files to it. If you pass it no arguments, it returns a string representing the current directory to which it saves session files. On my system, the following prints `/tmp`:

```
echo session_save_path();
```

A glance at my `/tmp` directory reveals a number of files with names like the following:

```
sess_fa963e3e49186764b0218e82d050de7b
sess_76cae8ac1231b11afa2c69935c11dd95
sess_bb50771a769c605ab77424d59c784ea0
```


Opening the file that matches the session ID I was allocated when I first ran Listing 12.2, I can see how the registered variables have been stored:

```
product1|s:17:"Sonic Screwdriver";product2|s:8:"HAL 2000";
```

When a value is placed in the `$_SESSION` superglobal, PHP writes the variable name and value to a file. This information can be read and the variables resurrected later—as you have already seen. After you add a variable to the `$_SESSION` superglobal, you can still change its value at any time during the execution of your script, but the altered value is not reflected in the global setting until you reassign the variable to the `$_SESSION` superglobal.

The example in Listing 12.3 demonstrates the process of adding variables to the `$_SESSION` superglobal. This example is not very flexible, however. Ideally, you should be able to register a varying number of values. You might want to let users pick products from a list, for example. In this case, you can use the `serialize()` function to store an array in your session.

Listing 12.5 creates a form that allows a user to choose multiple products. You use the session variables to create a rudimentary shopping cart.

LISTING 12.5 Adding an Array Variable to a Session Variable

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<head>
<title>Storing an array with a session</title>
</head>
<body>
<h1>Product Choice Page</h1>
<?php
if (isset($_POST['form_products'])) {
    if (!empty($_SESSION['products'])) {
        $products = array_unique(
        array_merge(unserialize($_SESSION['products']),
        $_POST['form_products']));
        $_SESSION['products'] = serialize($products);
    } else {
        $_SESSION['products'] = serialize($_POST['form_products']);
  }
   echo "<p>Your products have been registered!</p>";
}
?>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
<p><label for="form_products">Select some products:</label><br />
<select id="form_products" name="form_products[]" multiple="multiple"
size="3">
<option value="Sonic Screwdriver">Sonic Screwdriver</option>
<option value="Hal 2000">Hal 2000</option>
<option value="Tardis">Tardis</option>
<option value="ORAC">ORAC</option>
<option value="Transporter bracelet">Transporter bracelet</option>
</select></p>
<button type="submit" name="submit" value="choose">Submit Form</button>
</form>
<p><a href="session1.php">go to content page</a></p>
</body>
</html>
```

- The listing starts or resumes a session by calling session_start() on line 2. This call gives access to any previously set session variables. An HTML form begins on line 24 and, on line 26, creates a SELECT element named form_products[], which contains OPTION elements for a number of products.

- The block of PHP code beginning on line 11 tests for the presence of the $_POST[‘form_products’] array (line 12).
- If the variable is present, you can assume that the form has been submitted and information has already been stored in the $_SESSION superglobal.
- Line 12 tests for an array called $_SESSION[‘products’]. If the array exists, it was populated on a previous visit to this script, so the code merges it with the $_POST[‘form_products’] array, extracts the unique elements, and assigns the result back to the $products array (lines 14–16).
- Then the $products array is added to the $_SESSION superglobal on line 17. Line 35 contains a link to another script, which will demonstrate access to the products the user has chosen.

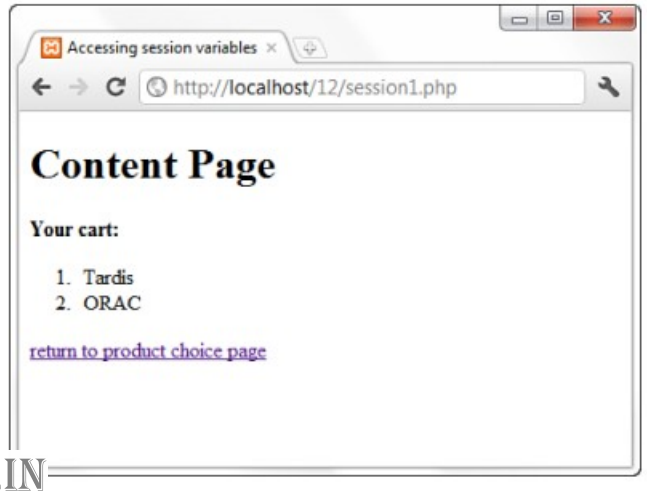
Accessing an Array of Session Variables

- This new script is created in Listing 12.6, but in the mean time you can save the code in Listing 12.5 as arraysession.php. Moving on to Listing 12.6, you see how to access the items stored in the session created in arraysession.php.

LISTING 12.6 Accessing Session Variables

```
 1: <?php
 2: session_start();
 3: ?>
 4: <!DOCTYPE html>
 5: <html>
 6: <head>
 7: <title>Accessing session variables</title>
 8: </head>
 9: <body>
10: <h1>Content Page</h1>
11: <?php
12: if (isset($_SESSION['products'])) {
13:     echo "<strong>Your cart:</strong><ol>";
14:     foreach (unserialize($_SESSION['products]) as $p) {
15:         echo "<li>".$p."</li>";
16:     }
17:     echo "</ol>";
18: }
19: ?>
20: <p><a href="arraysession.php">return to product choice page</a></p>
21: </body>
22: </html>
```

Once again, `session_start()` resumes the session on line 2. Line 12 tests for the presence of the `$_SESSION['products']` variable. If it exists, the variable is unserialized and looped through on lines 14–16, printing each of the user's chosen items to the browser. Figure 12.3 shows an example of the output.



Destroying Sessions and Unsetting Variables

You can use `session_destroy()` to end a session, erasing all session variables. The `session_destroy()` function requires no arguments. You should have an established session for this function to work as expected. The following code fragment resumes a session and abruptly destroys it:

```
session_start();
session_destroy();
```

When you move on to other pages that work with a session, the session you have destroyed will not be available to them, forcing them to initiate new sessions of their own. Any registered variables will be lost.

Questions

- Explain any six string handling functions used in PHP with example(6)
- How does a PHP array differ from an array in C? List the different ways to create an array in PHP with an example. Explain any 4 functions that deals with PHP array.(8)
- During the process of fetching a web page from a web server to a client browser, at what point does an embedded PHP script get executed. What are the two modes that the PHP processor operates in? Explain (6)
- Why is PHP considered to be dynamically typed? Distinguish between

implode and explode function in PHP with suitable examples.

17. (a) Write equivalent PHP statements corresponding to the following: **(8)**

i. Declare an associative array named "ages" to store the key-value pairs ("Alice", 30), ("Bob", 30), ("Harry", 35), ("Mary", 32).

ii. Modify the value associated with the key "Mary" to 28.

iii. Sort the array according to values maintaining the key-value relationships and print the sorted key-value pairs.

iv. The entry identified by the key "Bob"

(b) What are the uses of cookies in web pages? Describe syntax for setting cookies in PHP. How can you access and delete the cookie using setcookie() function? **(6)**

18. (a) Write a PHP form handling program to perform the user registration of any website with a minimum of 5 different fields and insert the data into a MySQL table after establishing necessary connections with the DB, **(8)**

(b) Design the HTML page which enters a given number and embed the PHP code to display a message indicating, whether the number is odd or even, when clicking on the 'CHECK NUMBER' button. **(6)**

- Describe how input from an HTML form is retrieved in a PHP program, with an example(3)
- Discuss the various steps for establishing PHP-MySQL connection with a MySQL db

Course Outcome 3 (CO3):

1. Write a PHP program to store the name and roll no of 10 students in an Associative Array and Use foreach loop to process the array and Perform asort, rsort and ksort in the array. Illustrate with suitable output data
2. Design an HTML page which enters a given number, write a PHP program to display a message indicating, whether the number is odd or even, when clicking on the submit button.
3. Write a PHP program to compute the sum of the positive integers up to 100 using do while.

Course Outcome 4 (CO4):

1. Write a PHP form handling program to verify the user authentication credentials of a web page using MySQL connection and store the userid value as a Session variable if the userid is valid.
2. Create a valid HTML document for yourself, including your name, address, and email address. Also add your college; your major and the course. Perform form handling in PHP and process the output using POST method.
3. Write an embedded PHP script which displays the factorial of all numbers from 1 to 10 in a table in the web page. The factorial should be calculated and returned from a function. The table headings should be "Number" and "Factorial"

- [PHP Sorting Arrays - sort(), rsort(), asort(), ksort(), arsort(), krsort() | jobtensor]

PHP& MySQL Integration-Connecting to MySQL with PHP .

- Making a Connection

The basic syntax for a connection to MySQL is as follows:

```
$mysqli = mysqli_connect("hostname", "username", "password", "database");
```

The value of `$mysqli` is the result of the function and is used in later functions for communicating with MySQL.

With sample values inserted, the connection code looks like this:

```
$mysqli = mysqli_connect("localhost", "joeuser", "somepass", "testDB");
```

Making a Connection

Listing 18.1 is a working example of a connection script. It creates a new connection in line 2 and then tests to see whether an error occurred. If an error occurred, line 5 prints an error message and uses the `mysqli_connect_error()` function to print the message. If no error occurs, line 8 prints a message that includes host information resulting from calling the `mysqli_get_host_info()` function.

LISTING 18.1 A Simple Connection Script

```
1:  <?php
2:  $mysqli = new mysqli("localhost", "joeuser", "somepass", "testDB");
3:
4:  if (mysqli_connect_errno()) {
5:     printf("Connect failed: %s\n", mysqli_connect_error());
6:     exit();
7:  } else {
8:      printf("Host information: %s\n", mysqli_get_host_info($mysqli));
9:  }
10: ?>
```

Save this script as `mysqlconnect.php` and place it in the document area of your web server. Access the script with your web browser and you will see something like the following, if the connection was successful:

```
Host information: localhost via TCP/IP
```

Making a Connection

You might also see this:

```
Host information: Localhost via UNIX socket
```

If the connection fails, an error message is printed. Line 5 generates an error via the `mysqli_connect_error()` function. An example is shown here:

```
Connect failed: Access denied for user 'joeuser'@'localhost' (using password:
  YES)
```

```
However, if the connection is successful, line 8 prints the output of
  mysqli_get_host_info(), such as examples above.
```

Although the connection closes when the script finishes its execution, it is a good practice to close the connection explicitly. You can see how to do this in line 9 of Listing 18.2, using the `mysqli_close()` function.

Modified connection

LISTING 18.2 The Modified Simple Connection Script

```
1:  <?php
2:  $mysqli = new mysqli("localhost", "joeuser", "somepass", "testDB");
3:
4:  if (mysqli_connect_errno()) {
5:      printf("Connect failed: %s\n", mysqli_connect_error());
6:      exit();
7:  } else {
8:    printf("Host information: %s\n", mysqli_get_host_info($mysqli));
9:    mysqli_close($mysqli);
10:  }
11:  ?>
```

Executing Queries

- The mysqli_query() function in PHP is used to send your SQL query to MySQL.

LISTING 18.3 A Script to Create a Table

```
1:  <?php
2:  $mysqli = mysqli_connect("localhost", "joeuser", "somepass", "testDB");
3:
4:  if (mysqli_connect_errno()) {
5:      printf("Connect failed: %s\n", mysqli_connect_error());
6:      exit();
7:  } else {
8:      $sql = "CREATE TABLE testTable
9:             (id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
10:            testField VARCHAR(75))";
11:     $res = mysqli_query($mysqli, $sql);
12:
13:     if ($res === TRUE) {
14:            echo "Table testTable successfully created.";
15:     } else {
16:         printf("Could not create table: %s\n", mysqli_error($mysqli));
17:     }
18:
19:     mysqli_close($mysqli);
20: }
21: ?>
```

- In lines 8–10, the text that makes up the SQL statement is assigned to the variable $sql. This is arbitrary, and you do not even need to place the content of your SQL query in a separate variable

Executing Queries

- The mysqli_query function returns a value of true or false, and this value is checked in the if…else statement beginning in line 13.
- If the value of $res is true, a success message is printed to the screen.
- If you access MySQL through the command-line interface to verify the creation of the testTable table, you will see the following output of DESCRIBE testTable:

```
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(11)     |      | PRI | NULL    | auto_increment |
| testField | varchar(75) | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
```

Executing Queries

- When mysqli_error() has been used in conjunction with the PHP die() function, which simply exits the script at the point at which it appears, the mysqli_error() function returns a helpful error message when you make a mistake.

Working with MySQL Data

- Inserting, updating, deleting, and retrieving data all revolve around the use of the mysqli_query() function to execute the basic SQL queries.
- SQL Injection: In the types of dynamic websites or web-based applications that you are likely to build, you will most often be INSERTing data into a table or SELECTing from a table based on user input from a form or other process.
- If you do not pay attention to this user input and sanitize it before using it in your queries, you are vulnerable to SQL injection

Working with MySQL Data

- SQL injection happens when nefarious individuals take the opportunity to type full or partial SQL queries in your form fields, with the assumption that when the script processes these queries, security will be breached and data potentially exposed.

Example-SQL Injection

```
$username = $_POST["username"];
$password = $_POST["password"];

$sql = "SELECT * FROM Users WHERE username = \"" . $username . "\" AND password = \"" . $
```

If user inputs username as `root` and password as `pass`, the SQL will interpret,

```
SELECT * FROM Users WHERE username = "root" AND password = "pass"
```


This code snippet looks fine when user inputs correct username and password.

What if the user inputs username as `invalid_user" OR "1"="1` and password as `invalid_pass" OR "1"="1`? Let's take a look at how SQL interprets.

```
SELECT * FROM Users WHERE username = "invalid_user" OR "1"="1" AND password = "invalid_p
```

Since, `"1"="1"` is always true, no matter what the username and password user enters, SQL will fetch all the users from the database.

Performing CREATE, DELETE, INSERT operations on MySQL table from PHP Program

- Refer pdf shared

Performing SELECT and UPDATE operations on MySQL table from PHP Program.

Inserting Data with PHP

LISTING 18.4 A Script to Insert a Record

```
1:  <?php
2:  $mysqli = mysqli_connect("localhost", "joeuser", "somepass", "testDB");
3:
4:  if (mysqli_connect_errno()) {
5:      printf("Connect failed: %s\n", mysqli_connect_error());
6:      exit();
7:  } else {
8:      $sql = "INSERT INTO testTable (testField) VALUES ('some value')";
9:      $res = mysqli_query($mysqli, $sql);
10:
11:      if ($res === TRUE) {
12:             echo "A record has been inserted.";
13:      } else {
14:          printf("Could not insert record: %s\n", mysqli_error($mysqli));
15:      }
16:
17:      mysqli_close($mysqli);
18:  }
19:  ?>
```


The only change between this script—for record insertion—and the script in Listing 18.3 for table creation is the SQL query stored in the `$sql` variable on line 8 and text modifications on lines 12 and 14. The connection code and the structure for issuing

Inserting Data with PHP

a query remain the same. In fact, most procedural code for accessing MySQL falls into this same type of code template.

Call this script `mysqlinsert.php` and place it on your web server. Running this script results in the addition of a row to the `testTable` table. To enter more records than the one shown in the script, you can either make a long list of hard-coded SQL statements and use `mysqli_query()` multiple times to execute these statements or you can create a form-based interface to the record addition script, which we do next.

To create the form for this script, you need only one field, because the `id` field can automatically increment. The action of the form is the name of the record-addition script; let's call it `insert.php`. Your HTML form might look something like Listing 18.5.

Inserting Data with PHP

LISTING 18.5 An Insert Form

```
1:  <!DOCTYPE html>
2:  <html>
3:  <head>
4:  <title>Record Insertion Form</title>
5:  </head>
6:  <body>
7:  <form action="insert.php" method="POST">
8:  <p><label for="testfield">Text to Add:</label><br/>
9:  <input type="text" id="testfield" name="testfield" size="30" /></p>
10: <button type="submit" name="submit" value="insert">Insert Record</button>
11: </form>
12: </body>
13: </html>
```

Save this file as `insert_form.html` and put it in the document root of your web server. Next, create the `insert.php` script shown in Listing 18.6. The value entered in the form replaces the hard-coded values in the SQL query with a variable called `$_POST['testfield']` (guarded against SQL injection, of course).

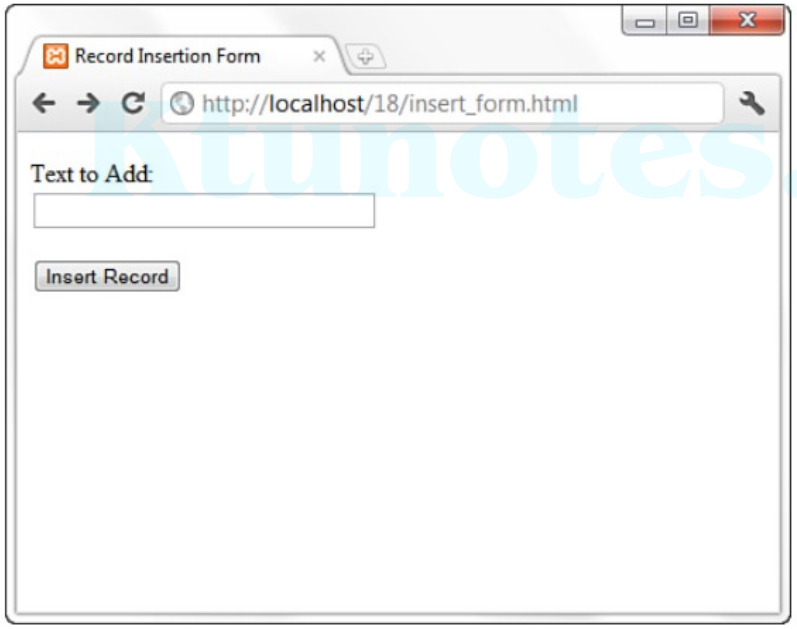
LISTING 18.6 An Insert Script Used with the Form

```
1:  <?php
2:  $mysqli = mysqli_connect("localhost", "joeuser", "somepass", "testDB");
3:
4:  if (mysqli_connect_errno()) {
5:      printf("Connect failed: %s\n", mysqli_connect_error());
6:      exit();
7:  } else {
8:      $clean_text = mysqli_real_escape_string($mysqli, $_POST['testfield']);
9:      $sql = "INSERT INTO testTable (testField)

10:             VALUES ('".$clean_text."')";
11:     $res = mysqli_query($mysqli, $sql);
12:
13:     if ($res === TRUE) {
14:             echo "A record has been inserted.";
15:     } else {
16:         printf("Could not insert record: %s\n", mysqli_error($mysqli));
17:     }
18:
19:     mysqli_close($mysqli);
20: }
21: ?>
```


The only changes between this script and the script in Listing 18.4 is line 8, where the form input is sanitized to avoid SQL injection, and in line 10, where we use the sanitized string `$clean_text` in place of the hard-coded text string from the previous example. To sanitize the input, we use the `mysqli_real_escape_string()` function; this function requires that a connection has already been made, and so it is placed in this position within the `else` portion of the `if…else` statement.

Inserting Data with PHP

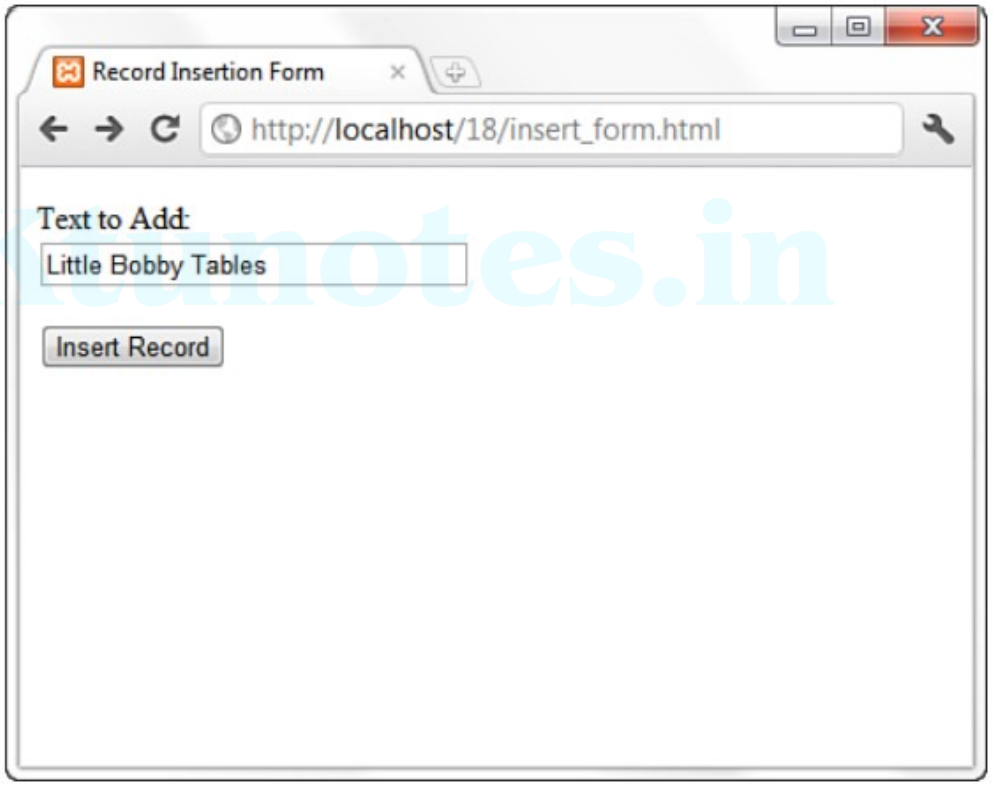
Save the script as `insert.php` and put it in the document root of your web server. In your web browser, access the HTML form that you created. It should look something like Figure 18.1.



Enter a string in the Text to Add field, as shown in Figure 18.2.

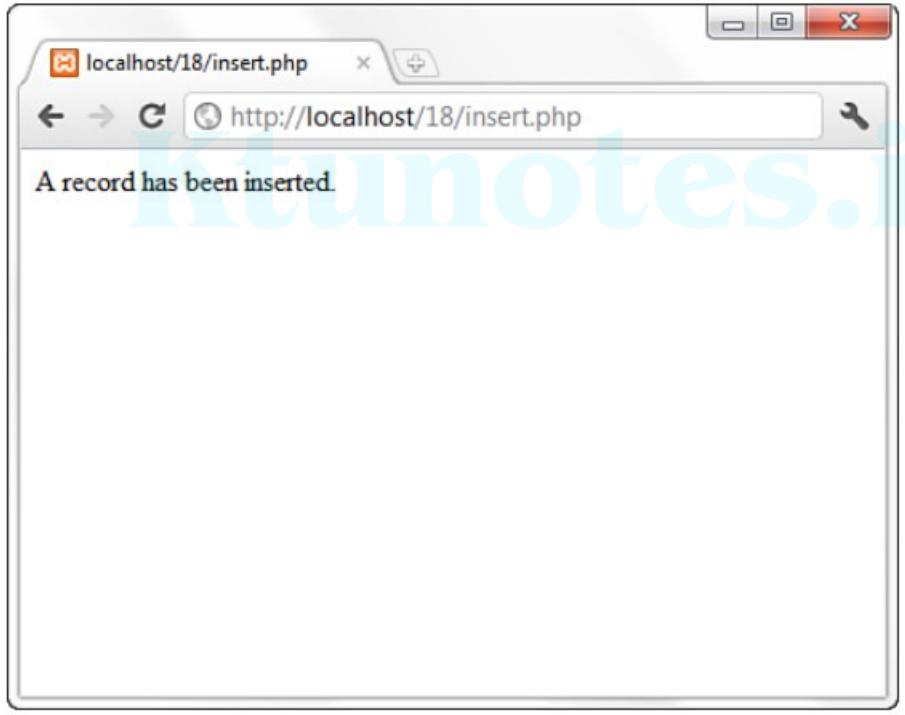
Inserting Data with PHP

Text typed in the form field



Inserting Data with PHP

- The record has been successfully added.



Inserting Data with PHP

To verify the work that has been done with PHP, you can use the MySQL command line interface to view the records in the table using a

SELECT query: SELECT * FROM testTable;

The output should be as follows:

```
+----+---------------------+
| id | testField           |
+----+---------------------+
|  1 | some value          |
|  2 | Little Bobby Tables |
+----+---------------------+
2 rows in set (0.00 sec)
```

Retrieving Data with PHP

- Because you have a few rows in your testTable table, you can write a PHP script to retrieve that data.
- We write a script that issues a SELECT query but doesn't overwhelm you with result data. Let's just get the number of rows. To do this, use the mysqli_num_rows() function.

LISTING 18.7 A Script to Retrieve Data

```
1:  <?php
2:  $mysqli = mysqli_connect("localhost", "joeuser", "somepass", "testDB");
3:
4:  if (mysqli_connect_errno()) {
5:      printf("Connect failed: %s\n", mysqli_connect_error());
6:      exit();
7:  } else {
8:      $sql = "SELECT * FROM testTable";
9:      $res = mysqli_query($mysqli, $sql);
10:
11:      if ($res) {
12:          $number_of_rows = mysqli_num_rows($res);
13:          printf("Result set has %d rows.\n", $number_of_rows);
14:      } else {
15:          printf("Could not retrieve records: %s\n", mysqli_error($mysqli));
16:      }
17:
18:      mysqli_free_result($res);
19:      mysqli_close($mysqli);
20:  }
21:  ?>
```

Retrieving Data with PHP

- Save this script as count.php, place it in your web server document directory, and access it through your web browser.
- Result set has 4 rows.

Line 12 uses the `mysqli_num_rows()` function to retrieve the number of rows in the resultset (`$res`), and it places the value in a variable called `$number_of_rows`. Line 13 prints this number to your browser. The number should be equal to the number of records you inserted during testing.

Retrieving Data with PHP

There's a new function in this listing that was not in previous listings. Line 18 shows the use of the `mysqli_free_result()` function. Using `mysqli_free_result()` before closing the connection with `mysqli_close()` ensures that all memory associated with the query and result is freed for use by other scripts.

Retrieving Data with PHP

- Easiest method is to retrieve each row from the table as an array
- You use a while statement to go through each record in the resultset, placing the values of each field into a specific variable and then displaying the results onscreen.
- The syntax of mysqli_fetch_array() is as follows:

```
$newArray = mysqli_fetch_array($result_set);
```

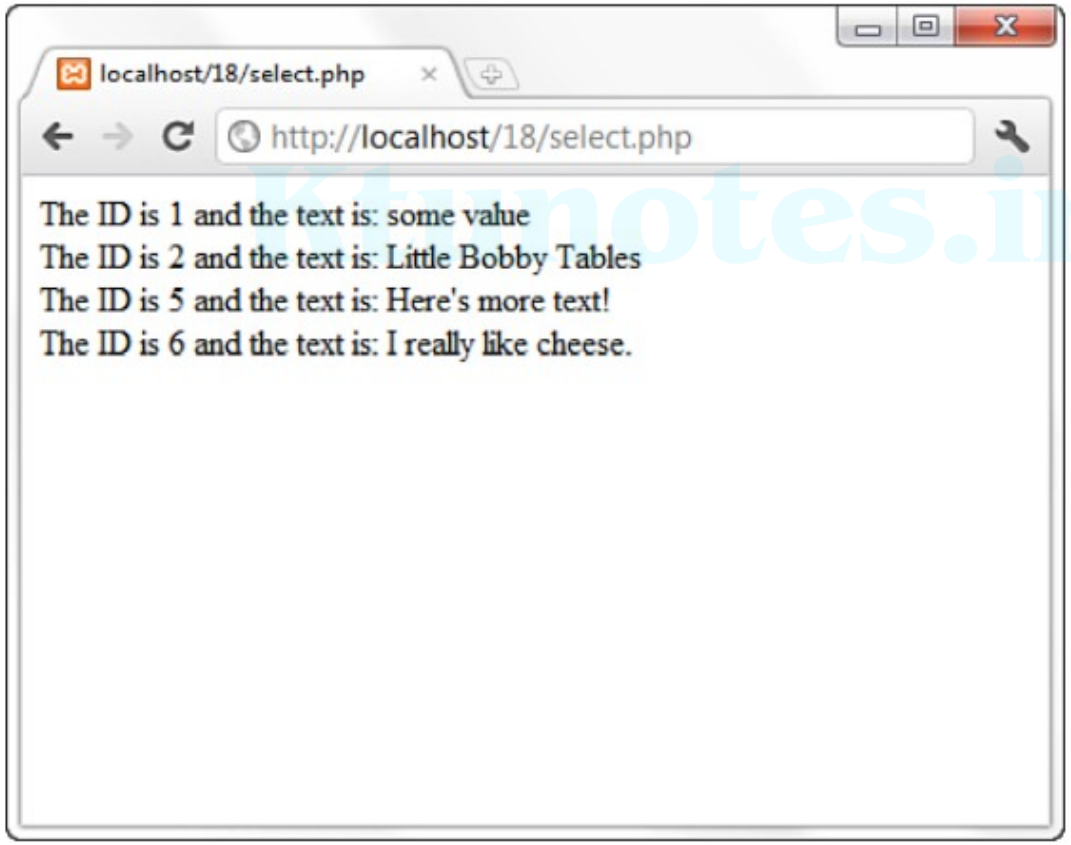
Retrieving Data with PHP

LISTING 18.8 A Script to Retrieve Data and Display Results

```
<?php
$mysqli = mysqli_connect("localhost", "joeuser", "somepass", "testDB");

if (mysqli_connect_errno()) {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
} else {
    $sql = "SELECT * FROM testTable";
    $res = mysqli_query($mysqli, $sql);

     if ($res) {
         while ($newArray = mysqli_fetch_array($res, MYSQLI_ASSOC)) {
             $id  = $newArray['id'];
             $testField = $newArray['testField'];
             echo "The ID is ".$id." and the text is: ".$testField."<br/>";
            }
     } else {
         printf("Could not retrieve records: %s\n", mysqli_error($mysqli));
     }

     mysqli_free_result($res);
     mysqli_close($mysqli);
 }
 ?>
```

Retrieving Data with PHP

- Selecting records from MySQL.



Building Dynamic Content in PHP application

- PHP can dynamically change the HTML5 it outputs based on a user's input

```
1  <!DOCTYPE html>
2
3  <!-- Fig. 19.20: dynamicForm.php -->
4  <!-- Dynamic form. -->
5  <html>
6     <head>
7        <meta charset = "utf-8">
8        <title>Registration Form</title>
9        <style type = "text/css">
10          p       { margin: 0px; }
11          .error  { color: red }
12          p.head  { font-weight: bold; margin-top: 10px; }
13          label   { width: 5em; float: left; }
14       </style>
15    </head>
16    <body>
17       <?php
18          // variables used in script
19          $fname = isset($_POST[ "fname" ]) ? $_POST[ "fname" ] : "";
20          $lname = isset($_POST[ "lname" ]) ? $_POST[ "lname" ] : "";
```

Fig. 19.20 | Dynamic form. (Part 1 of 5.)

```
21          $email = isset($_POST[ "email" ]) ? $_POST[ "email" ] : "";
22          $phone = isset($_POST[ "phone" ]) ? $_POST[ "phone" ] : "";
23          $book = isset($_POST[ "book" ]) ? $_POST[ "book" ] : "";
24          $os = isset($_POST[ "os" ]) ? $_POST[ "os" ] : "";
25          $iserror = false;
26          $formerrors =
27             array( "fnameerror" => false, "lnameerror" => false,
28                "emailerror" => false, "phoneerror" => false );
29
30          // array of book titles
31          $booklist = array( "Internet and WWW How to Program",
32             "C++ How to Program", "Java How to Program",
33             "Visual Basic How to Program" );
34
35          // array of possible operating systems
36          $systemlist = array( "Windows", "Mac OS X", "Linux", "Other" );
37
38          // array of name values for the text input fields
39          $inputlist = array( "fname" => "First Name",
40             "lname" => "Last Name", "email" => "Email",
41             "phone" => "Phone" );
42
43          // ensure that all fields have been filled in correctly
44          if ( isset( $_POST["submit"] ) )
45          {
46             if ( $fname == "" )
47             {
48                $formerrors[ "fnameerror" ] = true;
49                $iserror = true;
50             } // end if
51
52             if ( $lname == "" )
53             {
54                $formerrors[ "lnameerror" ] = true;
55                $iserror = true;
56             } // end if
57
58             if ( $email == "" )
59             {
60                $formerrors[ "emailerror" ] = true;
61                $iserror = true;
62             } // end if
63
64             if ( !preg_match( "/^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$/",
65                $phone ) )
66             {
67                $formerrors[ "phoneerror" ] = true;
68                $iserror = true;
69             } // end if
70
```

```
73              // build INSERT query
74              $query = "INSERT INTO contacts " .
75                 "( LastName, FirstName, Email, Phone, Book, OS ) " .
76                 "VALUES ( '$lname', '$fname', '$email', " .
77                 "'" . mysql_real_escape_string( $phone ) .
78                 "', '$book', '$os' )";
79
80              // Connect to MySQL
81              if ( !( $database = mysql_connect( "localhost",
82                 "iw3htp", "password" ) ) )
83                 die( "<p>Could not connect to database</p>" );
84
85              // open MailingList database
86              if ( !mysql_select_db( "MailingList", $database ) )
87                 die( "<p>Could not open MailingList database</p>" );
88
89              // execute query in MailingList database
90              if ( !( $result = mysql_query( $query, $database ) ) )
91              {
92                 print( "<p>Could not execute query!</p>" );
93                 die( mysql_error() );
94              } // end if
95
96              mysql_close( $database );
97
98              print( "<p>Hi $fname. Thank you for completing the survey.
99                 You have been added to the $book mailing list.</p>
100                <p class = 'head'>The following information has been
101                   saved in our database:</p>
102                <p>Name: $fname $lname</p>
103                <p>Email: $email</p>
104                <p>Phone: $phone</p>
105                <p>OS: $os</p>
106                <p><a href = 'formDatabase.php'>Click here to view
107                   entire database.</a></p>
108                <p class = 'head'>This is only a sample form.
109                   You have not been added to a mailing list.</p>
110                </body></html>" );
111             die(); // finish the page
112          } // end if
113       } // end if
114
115       print( "<h1>Sample Registration Form</h1>
116          <p>Please fill in all fields and click Register.</p>" );
117
118       if ( $iserror )
119       {
120          print( "<p class = 'error'>Fields with * need to be filled
121             in properly.</p>" );
122       } // end if
123
```

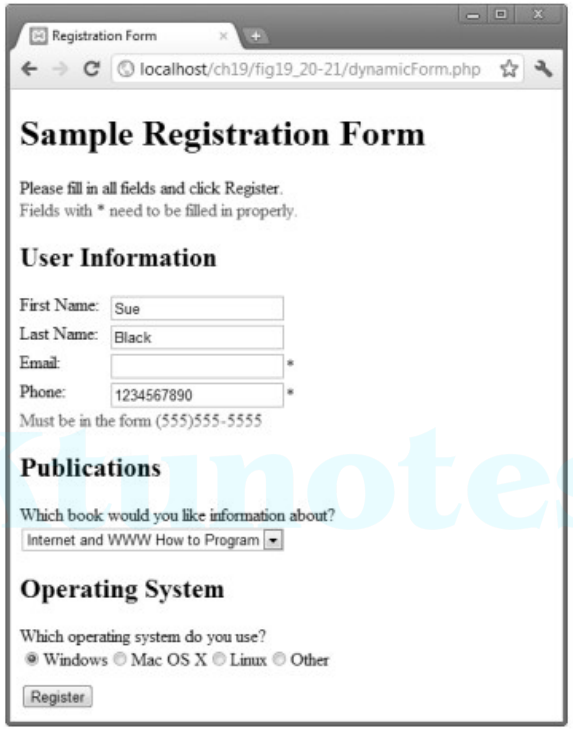

Fig. 19.20 | Dynamic form. (Part 3 of 3.)

```
124          print( "<!-- post form data to dynamicForm.php -->
125             <form method = 'post' action = 'dynamicForm.php'>
126             <h2>User Information</h2>
127
128             <!-- create four text boxes for user input -->" );
129          foreach ( $inputlist as $inputname => $inputalt )
130          {
131             print( "<div><label>$inputalt:</label><input type = 'text'
132                name = '$inputname' value = '" . $$inputname . "'>" );
133
134             if ( $formerrors[ ( $inputname )."error" ] == true )
135                print( "<span class = 'error'>*</span>" );
136
137             print( "</div>" );
138          } // end foreach
139
140          if ( $formerrors[ "phoneerror" ] )
141             print( "<p class = 'error'>Must be in the form
142                (555)555-5555" );
143
144          print( "<h2>Publications</h2>
145             <p>Which book would you like information about?</p>
146
147             <!-- create drop-down list containing book names -->
148             <select name = 'book'>" );
149
150          foreach ( $booklist as $currbook )
151          {
152             print( "<option" .
153                ($currbook == $book ? " selected>" : ">") .
154                $currbook . "</option>" );
155          } // end foreach
156
157          print( "</select>
158             <h2>Operating System</h2>
159             <p>Which operating system do you use?</p>
160
161             <!-- create five radio buttons -->" );
162
163          $counter = 0;
164
165          foreach ( $systemlist as $currsystem )
166          {
167             print( "<input type = 'radio' name = 'os'
168                value = '$currsystem' " );
169
170             if ( ( !$os && $counter == 0 ) || ( $currsystem == $os ) )
171                print( "checked" );
172
173             print( ">$currsystem" );
174             ++$counter;
175          } // end foreach
176
```


Fig. 19.20 | Dynamic form. (Part 4 of 5.)

```
177          print( "<!-- create a submit button -->
178             <p class = 'head'><input type = 'submit' name = 'submit'
179             value = 'Register'></p></form></body></html>" );
180    ?><!-- end PHP script -->
```

a) Registration form after it was submitted with a missing field and an incorrectly formatted phone number



b) Confirmation page displayed after the user properly fills in the form and the information is stored in the database

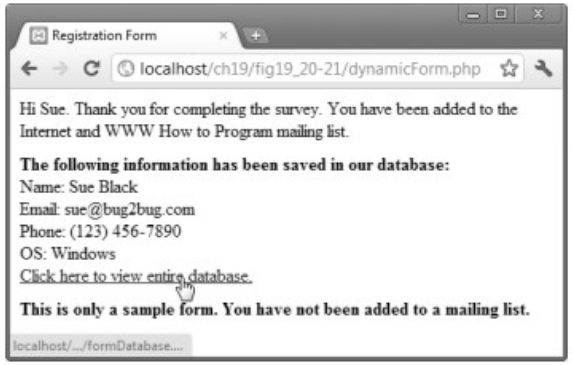


Fig. 19.20 | Dynam

Variables

- Lines 19–28 create variables that are used throughout the script to fill in form fields and check for errors.
- Lines 19–24 use the isset function to determine whether the $_POST array contains keys representing the various form fields.
- These keys exist only after the form is submitted.
- If function isset returns true, then the form has been submitted and we assign the value for each key to a variable. Otherwise, we assign the empty string to each variable.

Arrays

- Lines 31–41 create three arrays, $booklist, $systemlist and $inputlist, that are used to dynamically create the form’s input fields.
- We specify that the form created in this document is self-submitting (i.e., it posts to itself) by setting the action to the script 'dynamicForm.php' in line 125.
- Line 44 uses function isset to determine whether the Register button has been pressed, in which case the $_POST array will contain the key "submit" (the name of the button in the form).
- If it has, each of the text input fields’ values is validated. If an error is detected (e.g., a text field is blank or the phone number is improperly formatted), the corresponding entry in array $formerrors is set to true and variable $iserror is set to true.
- If the Register button has not been pressed, we skip ahead to line 115.

Dynamically Creating the Form

- Line 71 determines whether any errors were detected. If $iserror is false (i.e., there were no input errors)
- lines 74–111 display the page indicating that the form was submitted successfully—we'll say more about these lines later.
- If $iserror is true, lines 74–111 are skipped, and the code from lines 115–179 executes.
- These lines include a series of print statements and conditionals to output the form,(19.20 a)

Dynamically Creating the Form

- Lines 129–138 iterate through each element in the $inputlist array.
- In line 132 the value of $$inputname is assigned to the text field's value attribute. If the form has not yet been submitted, this will be the empty string "".
- The notation $$variable specifies a variable variable, which allows the code to reference variables dynamically.
- You can use this expression to obtain the value of the variable whose name is equal to the value of $variable.
- PHP first determines the value of $variable, then appends this value to the leading $ to form the identifier of the variable you wish to reference dynamically. (The expression $ $variable can also be written as ${$variable} to convey this procedure.)
- For example, in lines 129–138, we use $$inputname to reference the value of each form-field variable.

Dynamically Creating the Form

- During the iteration of the loop, $inputname contains the name of one of the text input elements, such as "email".
- PHP replaces $inputname in the expression $$inputname with the string representing that element’s name forming the expression ${"email"}.
- The entire expression then evaluates to the value of the variable $email. Thus, the variable $email, which stores the value of the e-mail text field after the form has been submitted, is dynamically referenced.
- This dynamic variable reference is added to the string as the value of the input field (using the concatenation operator) to maintain data over multiple submissions of the form

Dynamically Creating the Form

- Lines 134–135 add a red asterisk next to the text input fields that were filled out incorrectly.
- Lines 140–142 display the phone number format instructions in red if the user entered an invalid phone number.
- Lines 150–155 and 165–175 generate options for the book drop-down list and operating-system radio buttons, respectively.
- In both cases, we ensure that the previously selected or checked element (if one exists) remains selected or checked over multiple attempts to correctly fill out the form.
- If any book was previously selected, line 153 adds selected to its option tag.
- Lines 170–171 select an operating system radio button under two conditions.
- If the form is begin displayed for the first time, the first radio button is selected.
- Otherwise, if the $currsystem variable's value matches what's stored in the $os variable (i.e., what was submitted as part of the form), that specific radio button is selected.

Inserting Data into the Database

- Inserting Data into the Database If the form has been filled out correctly, lines 74–95 place the form information in the MySQL database MailingList using an INSERT statement.
- Line 77 uses the function mysql_real_escape_string to insert a backslash (\) before any special characters in the passed string.
- We must use this function so that MySQL does not interpret the parentheses in the phone number as having a special meaning aside from being part of a value to insert into the database.
- Lines 98–110 generate the web page indicating a successful form submission, which also provides a link to formDatabase.php (Fig. 19.21).

Displaying the Database's Contents

- The script in Fig. 19.21 displays the contents of the MailingList database.

```
<!DOCTYPE html>

<!-- Fig. 19.21: formDatabase.php -->
<!-- Displaying the MailingList database. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Search Results</title>
      <style type = "text/css">
         table  { background-color: lightblue;
                  border: 1px solid gray;
                  border-collapse: collapse; }
         th, td { padding: 5px; border: 1px solid gray; }
         tr:nth-child(even) { background-color: white; }
         tr:first-child { background-color: lightgreen; }
      </style>
   </head>
   <body>
      <?php
         // build SELECT query
         $query = "SELECT * FROM contacts";

         // Connect to MySQL
         if ( !( $database = mysql_connect( "localhost",
         "iw3htp", "password" ) ) )
            die( "<p>Could not connect to database</p></body></html>" );

         // open MailingList database
         if ( !mysql_select_db( "MailingList", $database ) )
            die( "<p>Could not open MailingList database</p>
               </body></html>" );

         // query MailingList database
         if ( !( $result = mysql_query( $query, $database ) ) )
         {
            print( "<p>Could not execute query!</p>" );
            die( mysql_error() . "</body></html>" );
         } // end if
      ?><!-- end PHP script -->

      <h1>Mailing List Contacts</h1>
      <table>
         <caption>Contacts stored in the database</caption>
         <tr>
            <th>ID</th>
            <th>Last Name</th>
            <th>First Name</th>
            <th>E-mail Address</th>
            <th>Phone Number</th>
            <th>Book</th>
            <th>Operating System</th>
```

Displaying the Database's Contents

```
53          <?php
54             // fetch each record in result set
55             for ( $counter = 0; $row = mysql_fetch_row( $result );
56                ++$counter )
57             {
58                // build table to display results
59                print( "<tr>" );
60
61                foreach ( $row as $key => $value )
62                   print( "<td>$value</td>" );
63
64                print( "</tr>" );
65             } // end for
66
67             mysql_close( $database );
68          ?><!-- end PHP script -->
69       </table>
70    </body>
71 </html>
```

Fig. 19.21 | Displaying the MailingList database. (Part 2 of 2.)