Key Skills & Concepts

- Creating Arrays
- Working with Arrays
- Looping Through Arrays
- Sorting Arrays
- Navigating in Arrays
- Converting Arrays To and From Strings
- Joining and Splitting Arrays
- Comparing Arrays
- Handling Multidimensional Arrays
- Create a Form in HTML
- Add JavaScript to a Form
- Accepting and Filing Form Data in PHP

This chapter displays the power of PHP as a full-fledged programming language. In Chapters 5 and 6, I briefly introduced and discussed arrays, but these powerful elements are the backbone of PHP, and there is a lot more we need to discuss about them. To these, we'll add a discussion of forms and form handling.

Using Arrays

Arrays are a means of grouping and handling several values with a single entity. Arrays are common in programming languages, but PHP provides an uncommon number of tools to make extensive use of arrays. The values in arrays are generally variables, either strings or numbers, but they can also be constants. Arrays have a single name that stores multiple values using an index, or *key*, to identify individual values. The key can be either an integer or a unique string.

Creating Arrays

Arrays can be created either through assignment, as you do a variable, or through the `array()` function, as you saw in Chapter 5. For example, here are four methods of creating a four-element array of cars:

- Method 1: individual integer assignment

```
$cars[0] = "Ford";
$cars[1] = "Chevy";
$cars[2] = "Honda";
$cars[3] = "BMW";
```

- Method 2: `array()` function integer assignment

```
$cars = array(0 => "Ford", 1 => "Chevy", 2 => "Honda", 3 => "BMW");
```

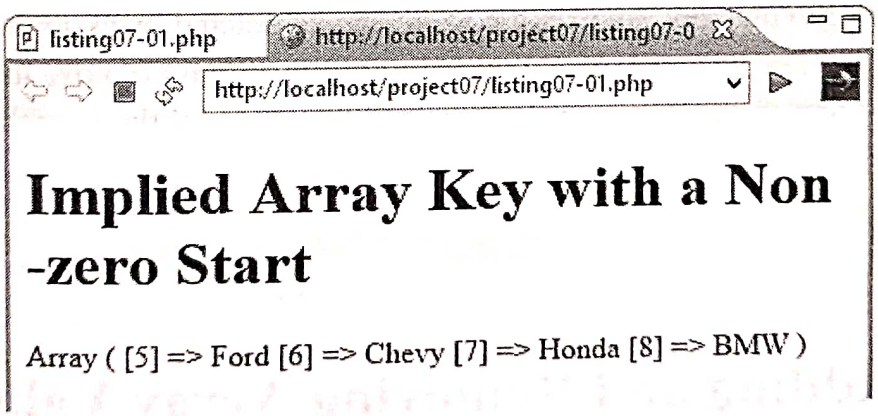
- Method 3: `array()` function-implied integer assignment

```
$cars = array ("Ford", "Chevy", "Honda", "BMW");
```

- Method 4: `array()` function-associative assignment

```
$cars = array ("Ed" => "Ford", "Sue" => "Chevy", "Kate" => "Honda", "Bob" => "BMW");
```

The first three methods produce the exact same arrays. In the third method, the numeric key is implied with the default start at 0. You can start at 1 or any other number by using the array operator =>. For example, Listing 7-1 produces the array shown next. The fourth method associates a string key with a value, which works just as well as an integer, and may be more meaningful.



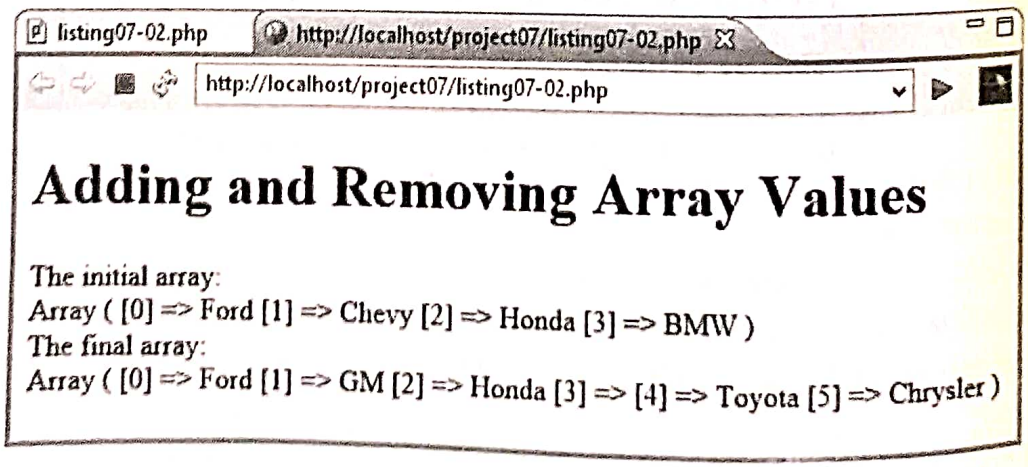
Listing 7-1 Implied Array Key with a Non-zero Start

```
<html>
    <head>
        <title>Listing 7-1</title>
    </head>
    <body>
        <h1>Implied Array Key with a Non-zero Start</h1>
        <?php
            $cars = array(
                5 => "Ford", "Chevy", "Honda", "BMW"
                );
            print_r($cars);
        ?>
    </body>
</html>
```

Working with Arrays

Once you have arrays, PHP provides a number of ways to change and work with them, including adding and removing values, using loops, sorting, navigating in them, converting to and from strings, comparing, and using array operators. There are also many more array functions than discussed in Chapter 5, and you can have multidimensional arrays.

PHP allows you to add, replace, and delete values within an array. You can add a value to an existing array either with a specific key or let PHP use the next implied numeric key. You replace a value by assigning the new value to an existing key, replacing the current value assigned to that key. An existing value can be deleted by assigning nothing ("") to the key, but the key will still exist. We will see ways to fully remove the item later in this chapter. Listing 7-2 shows one or more instances of each of these, with the results shown next.



Listing 7-2 Adding and Removing Array Values

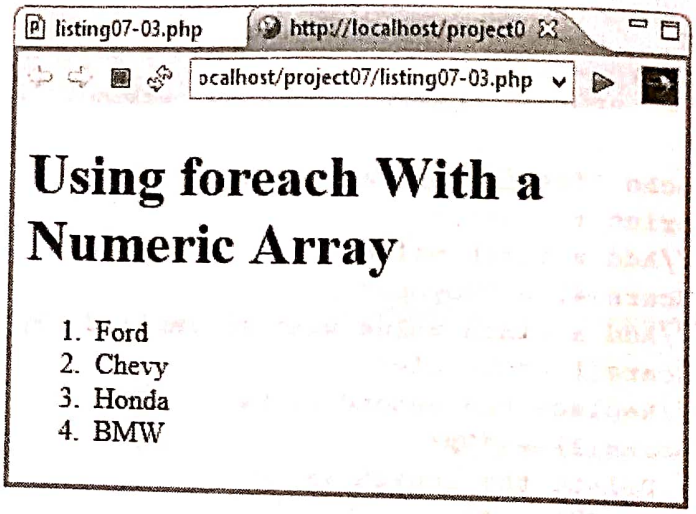
```
<html>
    <head>
        <title>Listing 7-2</title>
    </head>
    <body>
        <h1>Adding and Removing Array Values</h1>
        <?php
            $cars = array(
                "Ford", "Chevy", "Honda", "BMW"
                );o
            echo "The initial array: <br />" ;
            print_r($cars);
            //Add a fifth value
            $cars[4] = "Toyota" ;
            //Add a sixth value with an implied key
            $cars[] ="Chrysler" ;
            //Replace the second value
            $cars[1] = "GM" ;
            //Delete the fourth value
            $cars[3] = "" ;
            echo "<br /> The final array: <br />" ;
            print_r($cars);
            ?>
    </body>
</html>
```

Looping Through Arrays

As you have seen, the `print_r()` function automatically loops through an array and displays the `key=>value` pairs in an array. You can also do this using `for`, `while`, and `foreach` loops.

Using `foreach`

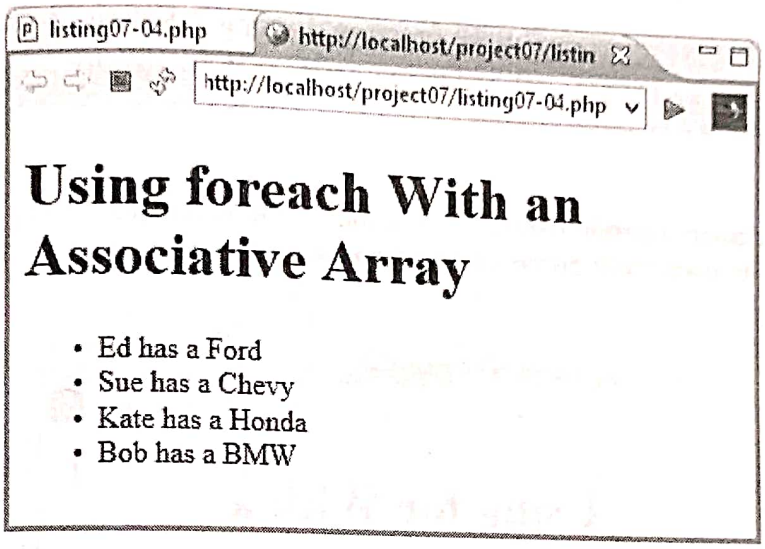
`foreach` is structured for use with arrays and uses the simple form `foreach ($arrayname as $valuename) ;` to successively place each of the values in `$arrayname` into `$valuename`, as shown in Listing 7-3 with an HTML ordered (numbered) list and displayed next (notice that even though the default numeric keys in an array start with 0, HTML ordered list starts at 1).



Listing 7-3 Using `foreach` with a Numeric Array

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-3</title>
    </head>
    <body>
        <h1>Using foreach With a Numeric Array</h1>
        <ol>
        <?php
            $cars = array (
               "Ford", "Chevy", "Honda", "BMW"
                );
                foreach ($cars as $item) {
                    echo "<li>$item" ;
                }
        ?>
        </ol>
    </body>
</html>
```

You can also use `foreach` with an associative array using the form `foreach ($arrayname as $keyname => $valuename) ;` as shown in Listing 7-4 with an HTML unordered (bulleted) list and displayed next.



Listing 7-4 Using `foreach` with an Associative Array

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-4</title>
    </head>
    <body>
        <h1>Using foreach With an Associative Array</h1>
        <ul>
        <?php
            $cars = array (
               "Ed" => "Ford", "Sue" => "Chevy",
                   "Kate" => "Honda", "Bob" => "BMW"
                );
                foreach ($cars as $key => $item) {
                    echo "<li>$key has a $item" ;
                }
        ?>
        </ul>
    </body>
</html>
```

NOTE

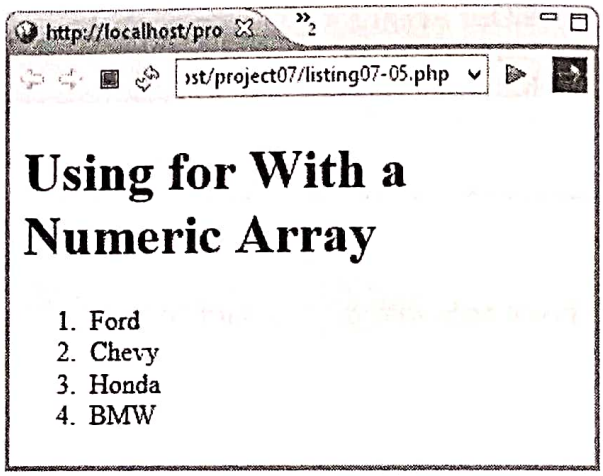
In Listings 7-3 and 7-4, the HTML ordered and unordered lists are for illustrative purposes and are not required.

Using `for`

Using a `for` loop with an array can use the `count()` function or its alias `sizeof()`, which does exactly the same thing: returns the number of elements in an array, with `count()` preferred since it is the original PHP function. Listing 7-5 shows how a `for` loop is used with an array and an HTML ordered list to produce these results:

NOTE

The technique demonstrated in Listing 7-5 with the `for` loop and the `count()` function will not work with associative arrays or with numeric arrays that begin with a number other than 0.

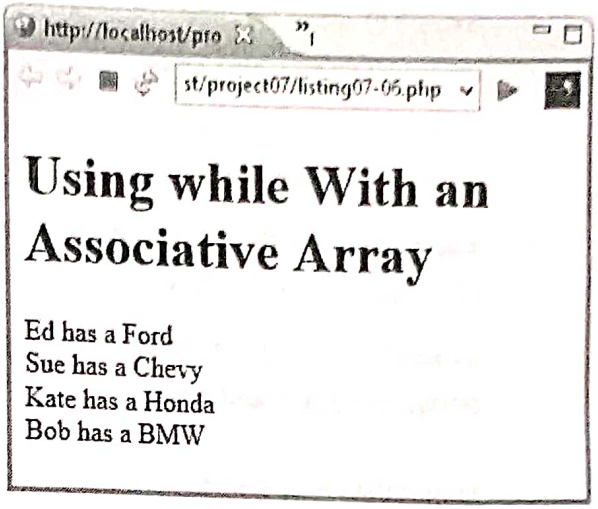


Listing 7-5 Using `for` with a Numeric Array

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-5</title>
    </head>
    <body>
        <h1>Using for With a Numeric Array</h1>
        <ol>
        <?php
            $cars = array (
               "Ford", "Chevy", "Honda", "BMW"
                );
                for ($i = 0; $i < count($cars); $i++) {
                    echo "<li>$cars[$i]" ;
                }
        ?>
        </ol>
    </body>
</html>
```

Using `while`

A `while` loop can use a construct very similar to a `for` loop, with the initialization of the index variable just before the `while` and its incrementing within the loop. You can simplify this using the `list()` function, which assigns variables as they are in an array, and the `each()` function, which returns the current `key => value` pair and advances the array pointer, as you see in Listing 7-6 with the results displayed next.



Listing 7-6 Using `while` with an Associative Array

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-6</title>
    </head>
    <body>
        <h1>Using while With an Associative Array</h1>
        <?php
            $cars = array (
               "Ed" => "Ford", "Sue" => "Chevy",
                   "Kate" => "Honda", "Bob" => "BMW"
                );
                while (list($key, $item) = each($cars)) {
                    echo "$key has a $item <br />" ;
                }
        ?>
    </body>
</html>
```

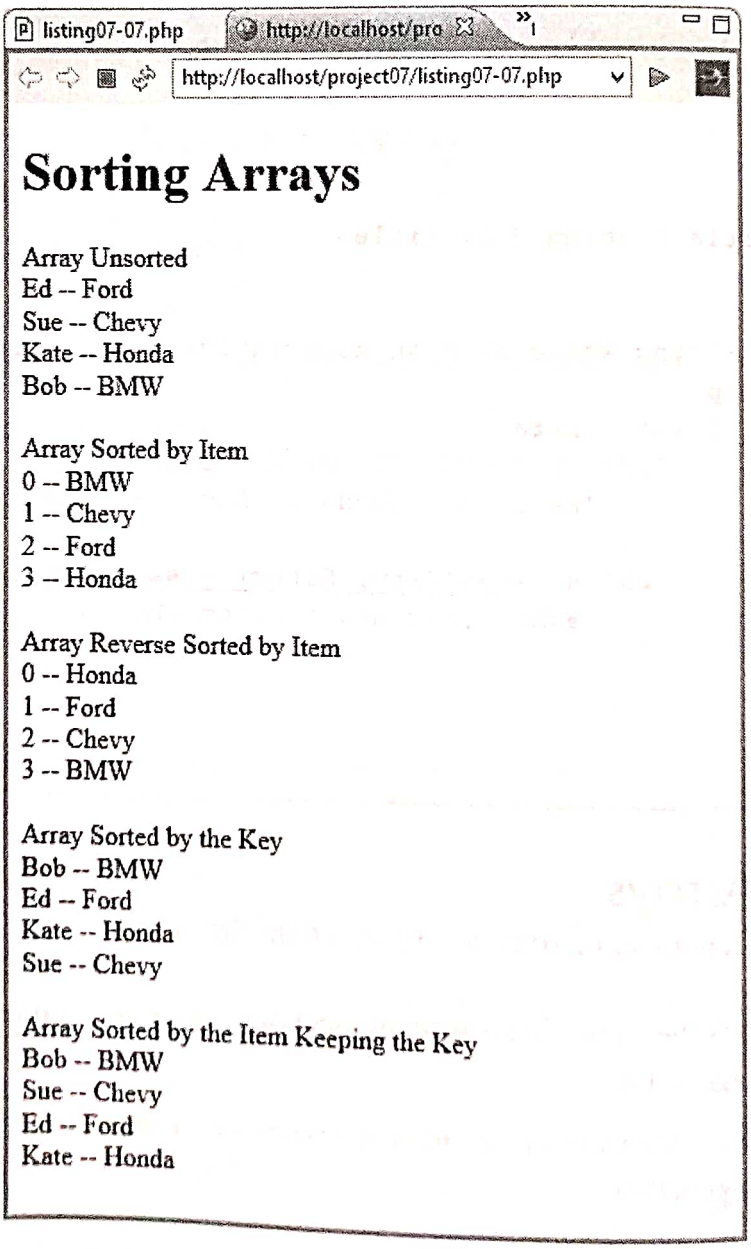
Sorting Arrays

PHP provides a number of array sorting functions, including

- `sort()` sorts the values in an array in ascending order and reassigns the keys as the default integers 0–9.
- `rsort()` sorts the values in an array in reverse order and reassigns the keys as the default integers 0–9.

- `ksort()` sorts the keys in an array in ascending order and maintains the association between values and keys.
- `krsort()` sorts the keys in an array in reverse order and maintains the association between values and keys.
- `asort()` sorts the values in an array in ascending order and maintains the association between values and keys.
- `arsort()` sorts the values in an array in reverse order and maintains the association between values and keys.

Using the array sort functions is a straightforward process, as you can see in Listing 7-7 with the results shown next:



Listing 7-7 Sorting Arrays

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-7</title>
    </head>
    <body>
        <h1>Sorting Arrays</h1>
        <?php
            $cars = array (
               "Ed" => "Ford", "Sue" => "Chevy",
                   "Kate" => "Honda", "Bob" => "BMW"
                );
                echo "Array Unsorted <br />" ;
                foreach ($cars as $key => $item) {
                    echo "$key -- $item <br />" ;
                }
            sort($cars);
                echo "<br /> Array Sorted by Item <br />" ;
                foreach ($cars as $key => $item) {
                    echo "$key -- $item <br />" ;
                }
            rsort($cars);
                echo "<br /> Array Reverse Sorted by Item <br />" ;
                foreach ($cars as $key => $item) {
                    echo "$key -- $item <br />" ;
                }
            $cars = array (
               "Ed" => "Ford", "Sue" => "Chevy",
                   "Kate" => "Honda", "Bob" => "BMW"
                );
            ksort($cars);
                echo "<br /> Array Sorted by the Key <br />" ;
                foreach ($cars as $key => $item) {
                    echo "$key -- $item <br />" ;
                }
            asort($cars);
                echo "<br /> Array Sorted by the Item Keeping
                    the Key <br />" ;
                foreach ($cars as $key => $item) {
                    echo "$key -- $item <br />" ;
                }
                ?>
    </body>
</html>
```

NOTE

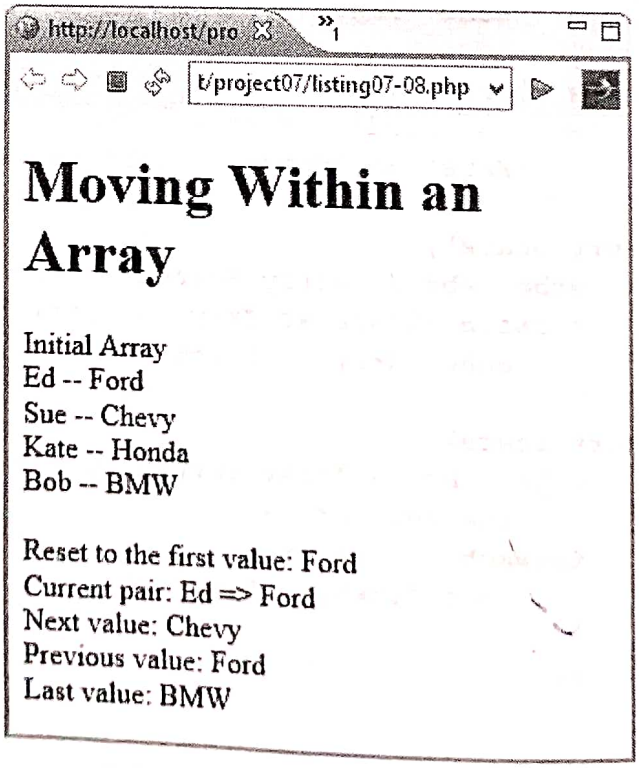
`sort` and `rsort` replace any associative keys with integers, and the array function has to be repeated to get them back.

Navigating in Arrays

In addition to looping, PHP provides several functions to allow you to move one element at a time through an array. These make use of an internal pointer that is initially set at the start of the session at the first element in the array. You can move this pointer within the array and reset it to the first element with these functions:

- `current()` returns the value currently at the pointer in an array and does not move the pointer.
- `key()` returns the key currently at the pointer in an array and does not move the pointer.
- `next()` moves the pointer one element forward in an array and returns the value.
- `prev()` moves the pointer one element back in an array and returns the value.
- `end()` moves the pointer to the last element in an array and returns the value.
- `reset()` moves the pointer to the first element in an array and returns the value.

Listing 7-8 demonstrates moving within an array with the results shown next.



Listing 7-8 Moving Within an Array

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-8</title>
    </head>
    <body>
        <h1>Moving Within an Array</h1>
        <?php
            $cars = array (
               "Ed" => "Ford", "Sue" => "Chevy",
                   "Kate" => "Honda", "Bob" => "BMW"
                );
                echo "Initial Array <br />" ;
                foreach ($cars as $key => $item) {
                    echo "$key -- $item <br />" ;
                }
                // Because foreach has moved the pointer
                  to the end, we must start with reset().
                echo "<br />Reset to the first value: ",
                    reset($cars), "<br />" ;
                echo "Current pair: ", key($cars), " => ",
                    current($cars), "<br />" ;
                echo "Next value: ", next($cars), "<br />" ;
                echo "Previous value: ", prev($cars), "<br />" ;
                echo "Last value: ", end($cars), "<br />" ;
        ?>
    </body>
</html>
```

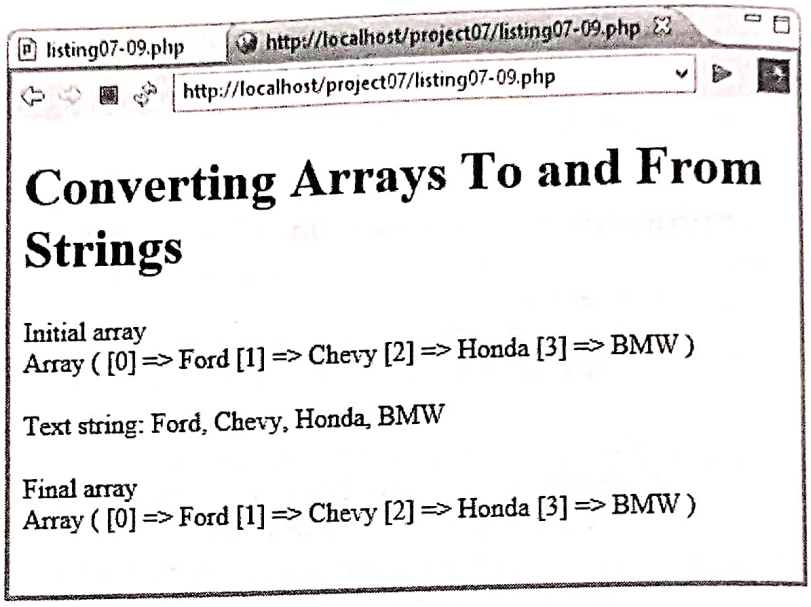
TIP

If you try to move beyond the last element in an array, the function will return `false`.

Converting Arrays To and From Strings

PHP provides two functions, **`implode()`** and **`explode()`**, to convert an array to a string and a string to an array, as shown in Listing 7-9 with the results shown next. Both **`implode()`** and **`explode()`** can have at least two parameters, a delimiter and either an array to be broken into a string in the case of **`implode()`**, or a string to be converted into an array in the case of **`explode()`**. The delimiter is a string that separates the elements of either the input or output string. For example, the delimiter used in Listing 7-9 with

both `implode()` and `explode()` is `", "`, a comma followed by a space. These are placed between the elements of the array.



Listing 7-9 Converting Arrays To and From Strings

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-9</title>
    </head>
    <body>
        <h1>Converting Arrays To and From Strings</h1>
        <?php
            $cars[0] = "Ford";
            $cars[1] = "Chevy";
            $cars[2] = "Honda";
            $cars[3] = "BMW";
            echo "Initial array <br />";
            print_r($cars);
            //Convert to a string
            $carlist = implode(", ", $cars) ;
            echo "<br /><br /> Text string:  ", $carlist ;
            //Convert back to an array
            $cars = explode(", ", $carlist) ;
            echo "<br /><br /> Final array <br />";
            print_r($cars);
            ?>
    </body>
</html>
```

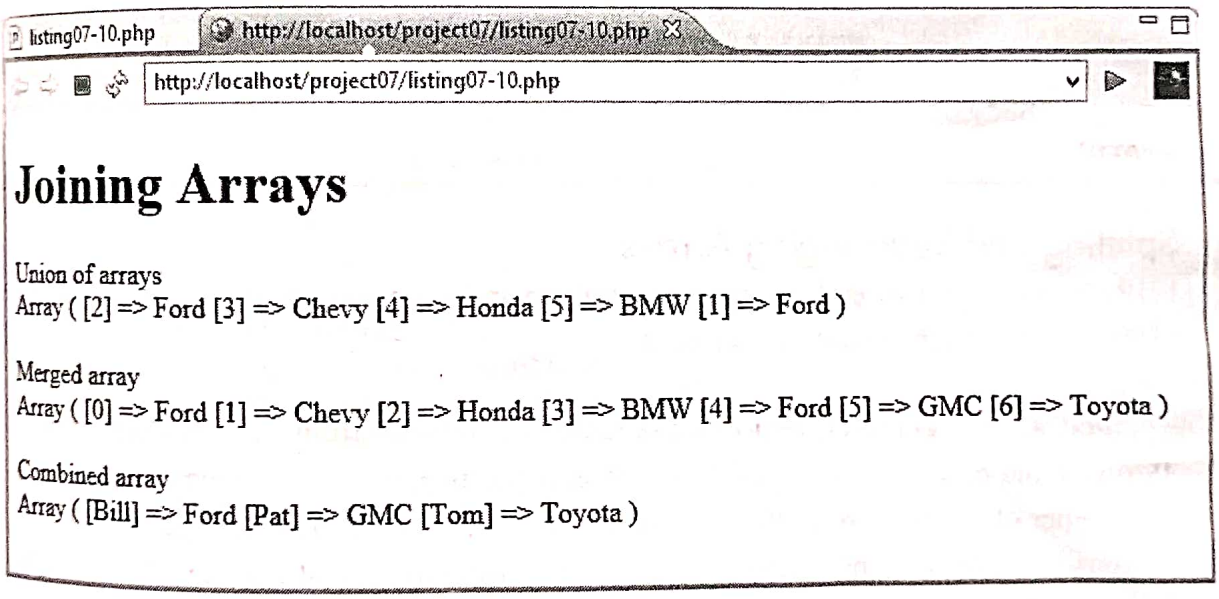
Joining and Splitting Arrays

As you work with arrays, you will want to put arrays together and take them apart. PHP has a number of ways to do both of these.

Joining Arrays

PHP provides several functions and an array operator to join two or more arrays, as shown in Listing 7-10 with the results shown next.

- **Union**, `$a + $b`, joins two arrays based on their unique keys, keeping the original keys.
- **Merge**, `array_merge()`, joins two arrays of values and renumbers their keys if they are numeric.
- **Combine**, `array_combine()`, joins an array of keys with an array of values, which must be of the same length as the keys.



Listing 7-10 Joining Arrays

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-10</title>
    </head>
    <body>
        <h1>Joining Arrays</h1>
```

```
<?php
        $cars[2] = "Ford";
        $cars[3] = "Chevy";
        $cars[4] = "Honda";
        $cars[5] = "BMW";
        $trucks[1] = "Ford";
        $trucks[3] = "GMC";
        $trucks[4] = "Toyota";
        $owners [1] = "Bill";
        $owners [2] = "Pat";
        $owners [3] = "Tom";
        $vehicles = $cars + $trucks;
        echo "Union of arrays <br />";
        print_r($vehicles);
        $vehicles = array_merge($cars, $trucks);
        echo "<br /><br />Merged array <br />";
        print_r($vehicles);
        $vehicles = array_combine($owners, $trucks);
        echo "<br /><br />Combined array <br />";
        print_r($vehicles);
        ?>
    </body>
</html>
```

Splitting and Rearranging Arrays

PHP also gives you several functions to break up and rearrange arrays, as you can see in Listing 7-11 with the results shown next.

- **Slice**, `array_slice()`, removes a sequence of elements from an array based on an offset and optionally a length. If the offset is positive, the starting element is that number of elements from the beginning of the array. If the offset is negative, the starting element is that number of elements in from the end of the array. If the length is positive and less than the total number of elements in the array, then the slice will include the length number of elements. If the length is positive and greater than the number in the array, the slice will include the remaining elements in the array. If the length is negative, then the slice will stop that number of elements from the end of the array. By default, a new set of numeric keys will be created for the resulting array, but you can include the `$preserve_keys` variable set to `TRUE` to change that behavior. Said another way, you can pass an optional fourth parameter set to `TRUE` to change that behavior.

- **Chunk**, `array_chunk()`, divides an array into chunks of equal size based on a size parameter, except that the last chunk may be less and produces a multidimensional array with the results. By default, a new set of numeric keys will be created for the resulting arrays, but you can include the `$preserve_keys` variable set to `TRUE` to change that behavior.
- **Splice**, `array_splice()`, removes a sequence of elements based on offset and length parameters as in `array_slice()`, and provides for the insertion of an optional replacement array where the sequence was removed. Keys are replaced by a new set.
- **Flip**, `array_flip()`, exchanges keys and values in an array. If there are multiple occurrences of a value in the original array, only the last one will be used and the others will be lost.

Breaking Up Arrays

```
The initial array:
Array ( [0] => Ford [1] => Chevy [2] => Honda [3] => BMW [4] => Toyota )

Slice out the middle three:
Array ( [0] => Chevy [1] => Honda [2] => BMW )

Slice out the last two:
Array ( [0] => BMW [1] => Toyota )

Divide into two car chunks:
Array ( [0] => Array ( [0] => Ford [1] => Chevy )
        [1] => Array ( [0] => Honda [1] => BMW )
        [2] => Array ( [0] => Toyota ) )

Splice in this array after removing the last two cars:
Array ( [0] => Fiat [1] => Mazda )
Giving this result:
Array ( [0] => Ford [1] => Chevy [2] => Honda [3] => Fiat [4] => Mazda )

Given this initial array:
Array ( [Ed] => Ford [Sue] => Chevy [Kate] => Ford [Bob] => BMW )
Flipping it produces this result:
Array ( [Ford] => Kate [Chevy] => Sue [BMW] => Bob )
```

Listing 7-11 Breaking Up Arrays

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-11</title>
    </head>
    <body>
        <h1>Breaking Up Arrays</h1>
        <?php
            $cars = array(
                "Ford", "Chevy", "Honda", "BMW", "Toyota"
                );
            echo "The initial array: <br />" ;
            print_r($cars);
            echo "<br /><br />Slice out the middle three:<br />";
            $mid_cars = array_slice($cars, 1 , 3) ;
            print_r($mid_cars);
            echo "<br /><br />Slice out the last two:<br />";
            $last_cars = array_slice($cars, -2 , 2) ;
            print_r($last_cars);
            echo "<br /><br />Divide into two car chunks:<br />";
            $car_chunks = array_chunk($cars, 2) ;
            print_r($car_chunks);
            echo "<br /><br />Splice in this array after removing
                the last two cars:<br />";
            $cars2 = array(
                "Fiat", "Mazda"
                );
            print_r($cars2);
            echo "<br />Giving this result:<br />";
            array_splice($cars, -2 , 2, $cars2) ;
            print_r($cars);
            $cars = array (
               "Ed" => "Ford", "Sue" => "Chevy", "Kate" => "Ford",
                    "Bob" => "BMW"
                );
            echo "<br /><br />Given this initial array:<br />";
            print_r($cars);
            echo "<br />Flipping it produces this result:<br />";
            $cars = array_flip($cars) ;
            print_r($cars);
        ?>
    </body>
</html>
```

Comparing Arrays

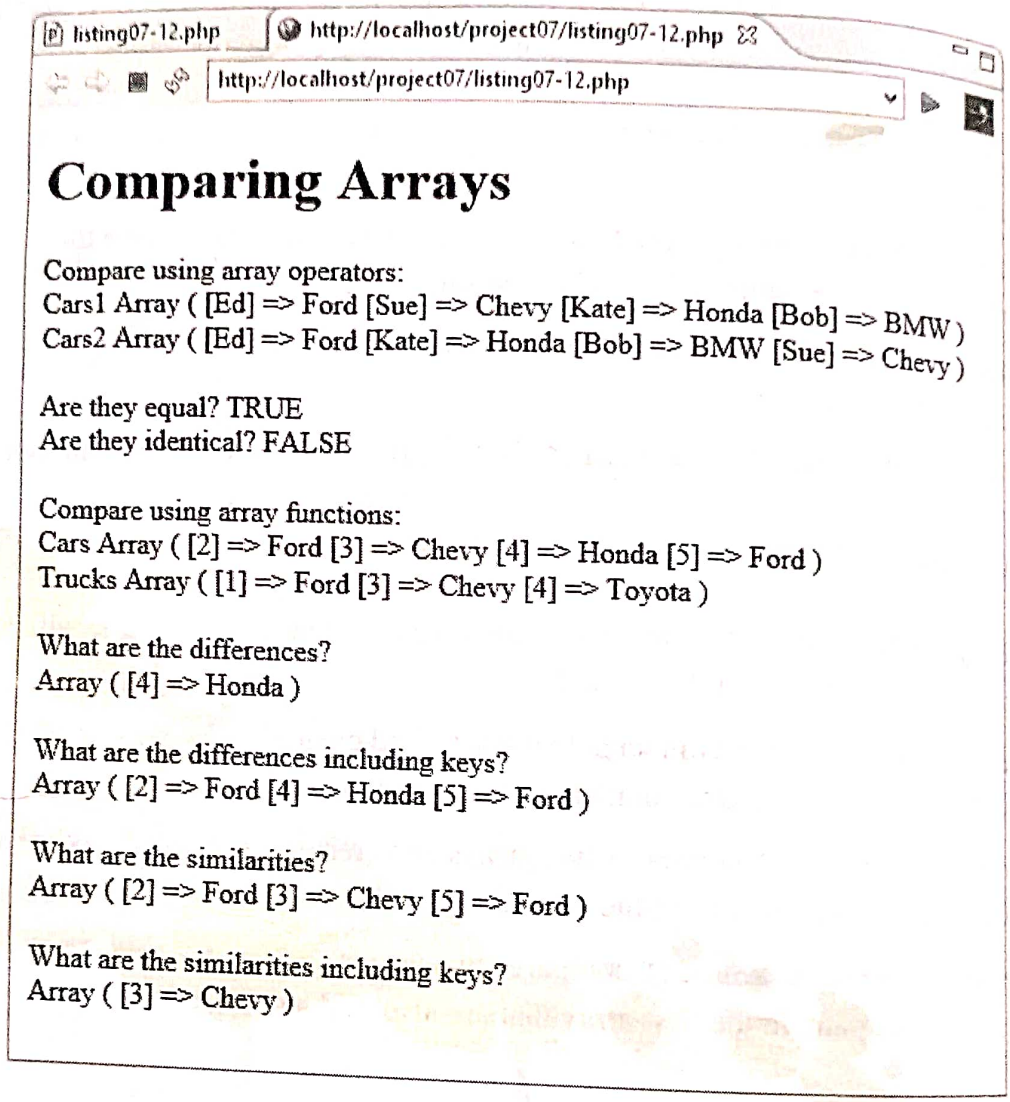
You can compare arrays in PHP with both array operators and functions. If you have two arrays, `$cars1` and `$cars2`, you can compare them with these operators:

- **Equality**, **`$cars1 == $cars2`**, is `TRUE` if `$cars1` and `$cars2` have the same key => value pairs.
- **Identity**, **`$cars1 === $cars2`**, is `TRUE` if `$cars1` and `$cars2` have the same key => value pairs, in the same order, and of the same type.
- **Inequality**, **`$cars1 != $cars2`** or **`$cars1 <> $cars2`**, is `TRUE` if `$cars1` and `$cars2` are not equal.
- **Non-identity**, **`$cars1 !== $cars2`**, is `TRUE` if `$cars1` and `$cars2` are not identical.

You can also compare arrays with these functions:

- `array_diff()` compares two arrays and creates a new array with the values in the first array that are not in the second.
- `array_diff_assoc()` compares two arrays and creates a new array with the key => value pairs in the first array that are not in the second.
- `array_intersect()` compares two arrays and creates a new array with the values in the first array that are also in the second.
- `array_intersect_assoc()` compares two arrays and creates a new array with the key => value pairs in the first array that are also in the second.

Listing 7-12 shows how the array operators and the array comparison functions can work, with the results shown next.



Listing 7-12 Comparing Arrays

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-12</title>
    </head>
    <body>
        <h1>Comparing Arrays</h1>
        <?php
            echo "Compare using array operators:<br />";
            $cars1 = array (
```

```
    "Ed" => "Ford", "Sue" => "Chevy", "Kate" =>
      "Honda", "Bob" => "BMW"
    );
echo "Cars1 ";
print_r($cars1);
$cars2 = array (
    "Ed" => "Ford", "Kate" => "Honda", "Bob" =>
      "BMW", "Sue" => "Chevy"
    );
echo "<br />Cars2 ";
print_r($cars2);
echo "<br /><br />Are they equal? ";
if( ($cars1 == $cars2) == TRUE) {
    echo "TRUE"; } else {echo "FALSE"; }
echo "<br />Are they identical? ";
if( ($cars1 === $cars2) == TRUE) {
    echo "TRUE"; } else {echo "FALSE"; }
echo "<br /><br /> Compare using array functions:<br />";
$cars[2] = "Ford";
$cars[3] = "Chevy";
$cars[4] = "Honda";
$cars[5] = "Ford";
echo "Cars ";
print_r($cars);
$trucks[1] = "Ford";
$trucks[3] = "Chevy";
$trucks[4] = "Toyota";
echo "<br />Trucks ";
print_r($trucks);
echo "<br /><br />What are the differences? <br />";
$vehicles = array_diff($cars, $trucks);
print_r($vehicles);
echo "<br /><br />What are the differences including
    keys? <br />";
$vehicles = array_diff_assoc($cars, $trucks);
print_r($vehicles);
echo "<br /><br />What are the similarities? <br />";
$vehicles = array_intersect($cars, $trucks);
print_r($vehicles);
echo "<br /><br />What are the similarities including
    keys? <br />";
$vehicles = array_intersect_assoc($cars, $trucks);
print_r($vehicles);
?>
</body>
</html>
```

Handling Multidimensional Arrays

PHP allows you to have arrays within arrays, or multidimensional arrays. For example, if you have an array of a person's contact information, like this:

```
$Sue = array ("email" => "sue@anisp.com", "phone" => "555-1234");
```

and then you have an array of contacts, like this:

```
$contacts = array ("Sue", "Tom", "Bob");
```

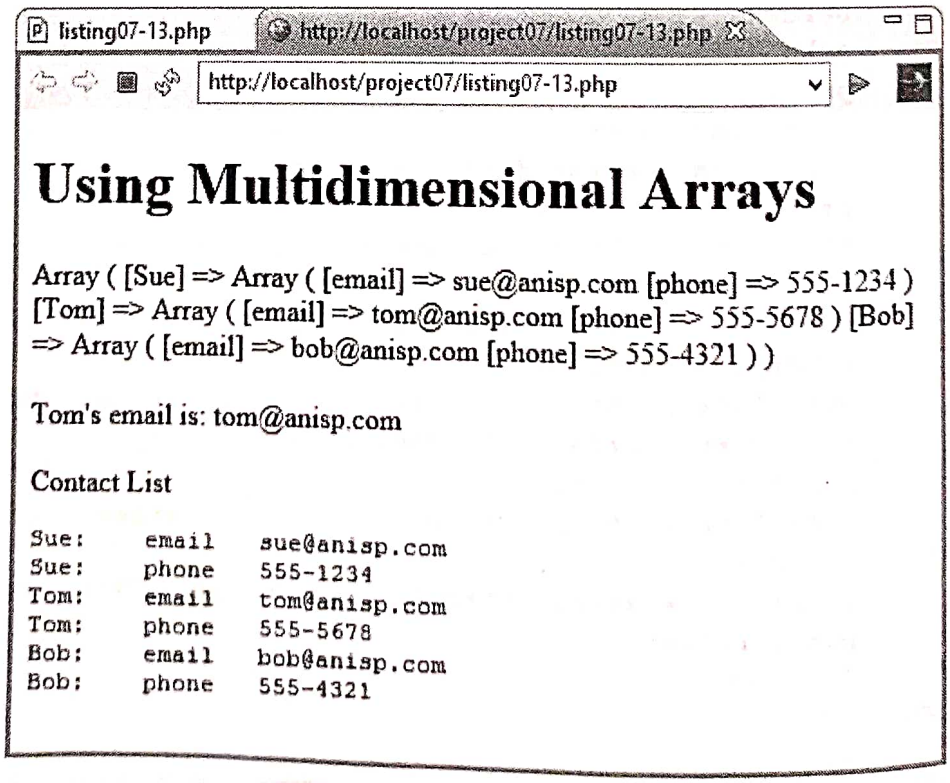
you can put them in a multidimensional array, like this:

```
$contacts=array(
  "Sue" => array( "email" => "sue@anisp.com", "phone" => "555-1234"),
  "Tom" => array( "email" => "tom@anisp.com", "phone" => "555-5678"),
  "Bob" => array( "email" => "bob@anisp.com", "phone" => "555-4321"));
```

You can refer to a value in a multidimensional array by using both keys, like this:

```
echo "Tom's email is: ", $contacts["Tom"] ["email"], "<br />";
```

You can also use looping functions to work with multidimensional arrays. Basically, you need a loop for each dimension of the array. Listing 7-13 shows using two `foreach()` functions to loop through the contacts array to display the contact information it contains, as shown next.



Listing 7-13 Using Multidimensional Arrays

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-13</title>
    </head>
    <body>
        <h1>Using Multidimensional Arrays</h1>
        <?php
            $contacts=array(
                "Sue"=>array("email"=>"sue@anisp.com",
                    "phone"=>"555-1234"),
                "Tom"=>array("email"=>"tom@anisp.com",
                    "phone"=>"555-5678"),
                "Bob"=>array("email"=>"bob@anisp.com",
                    "phone"=>"555-4321"));
            print_r($contacts);
            echo "<br /><br />Tom's email is: ", $contacts
               ["Tom"] ["email"], "<br />";
            echo "<br />Contact List<pre>";
            foreach ($contacts as $person => $means)
                foreach ($means as $key => $value )
                    echo "$person:\t$key\t$value<br />";
            echo "</pre>";
        ?>
    </body>
</html>
```

NOTE

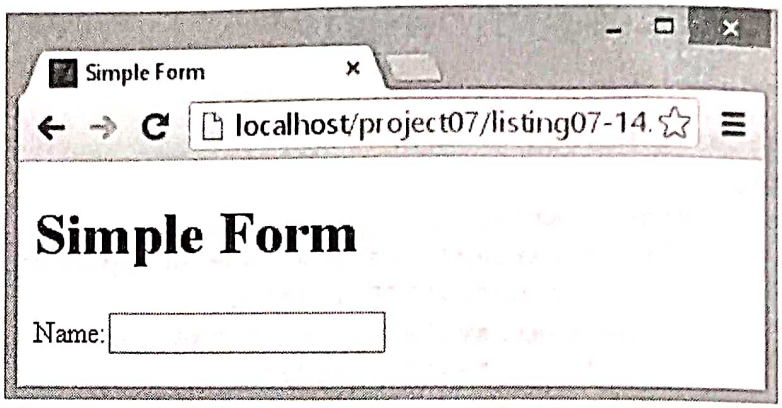
To use the tab escape character (`\t`) in the `echo` statement that displays the contact list, we must first turn on preformatted text with the HTML `<pre></pre>` tags to display text in a fixed-width font and preserve spaces, line breaks, and tabs.

Building and Handling Forms

The principle way to communicate with a user of a website is through forms. This process requires the combination of HTML, JavaScript, PHP, and eventually MySQL. In Chapters 2 and 4, you were introduced to the basic form creation and validating features of HTML and JavaScript. Here, we'll expand on those and add the form and file capabilities of PHP. In future chapters, you'll see how MySQL adds its functions to working with forms.

Create a Form in HTML

Creating a form in HTML is entering a pair of `<form> </form>` tags and then within those tags, entering the fields that you want with the `<input />` tag and `type` attribute. Listing 7-14 shows a simple form that is displayed next:

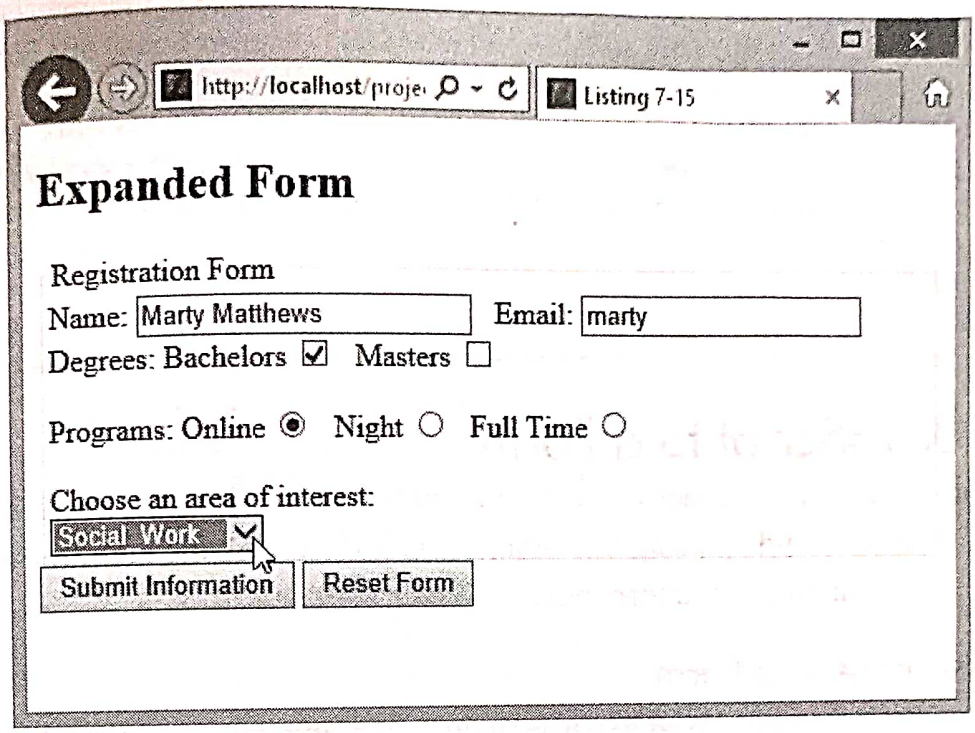


Listing 7-14 Simple Form

```
<!DOCTYPE html>
<html>
    <head>
        <title>Simple Form</title>
    </head>
    <body>
        <h1>Simple Form</h1>
        <form>
            Name:<input type="text" /><br />
        </form>
    </body>
</html>
```

There are, of course, many other `<input />` types and other form-related tags and attributes, as described in Chapter 2 and Table 2-8 and further enlarged upon in Chapter 4. For the sake of working with a form in PHP, enlarge the HTML form so you have more to work with by adding a second text box with an email address, a pair of radio buttons, a couple of check boxes, and a drop-down list. Also, name all of the fields so we can refer to them later, enclose the fields in a fieldset with a legend, and add submit and

reset buttons, as was done in Chapter 4. The resulting form is shown in Listing 7-15 and displayed next.



Listing 7-15 Expanded Form

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-15</title>
    </head>
    <body>
        <h2>Expanded Form</h2>
        <form name="regform">
            <fieldset>
                <legend>Registration Form</legend>
                Name: <input type="text" name="fullname" size =
                    "24"/>  
                Email: <input type="text" name="email" /><br />
                Degrees: Bachelors <input type="checkbox"
                    name="bachelors"/>   Masters <input
                    type="checkbox" name="masters"/><p></p>
                Programs: Online <input type="radio" name="online"/>
                      Night <input type="radio" name="night"/>
                      Full Time <input type="radio" name=
                    "fulltime"/><p></p>
                Choose areas of interest:<br />
```

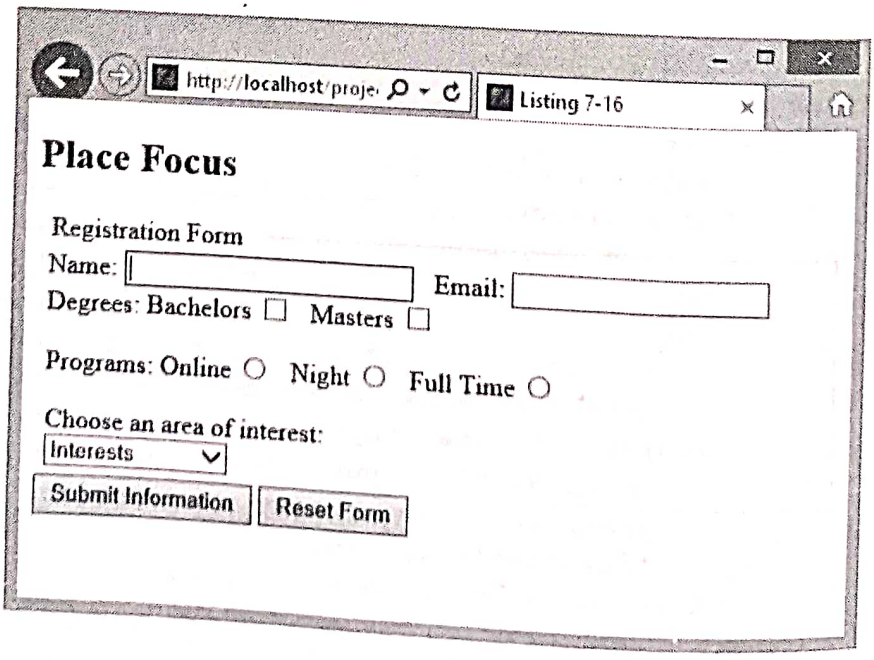
```
                <select name="Interests" >
                <option selected value="">Interests</option>
                <option value="socialwork">Social_Work</option>
                <option value="clinical">Clinical_Psych</option>
                <option value="education">Education</option>
                </select>
            </fieldset>
            <input type="submit" value="Submit Information">
            <input type="reset" value="Reset Form">
        </form>
    </body>
</html>
```

Add JavaScript to a Form

There are a number of elements that JavaScript brings to forms, as described in Chapter 4. These elements include placing the initial focus and recognizing and validating user input. We add those features to the form next.

Placing Focus in a Form

When a form is first displayed to a user, there is nothing inherent in the display that places the focus or insertion point (the cursor) into the form field to be first filled in. For example, in Listing 7-15 in the last section, the user must click in the Name text box to enter their name. Placing the focus in the text box can be done with JavaScript. You must identify that you will be using JavaScript in the `<head>` section and place an `onload` event handler in the `<body>` tag, as you see in Listing 7-16 (except for the changes in the `<head>` section and in the `<body>` tag, it is the same as Listing 7-15, so I only show the first few lines) as displayed next.



Listing 7-16 Place Focus

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-16</title>
        <script type= "text/javascript"></script>
    </head>
    <body onload = "document.regform.fullname.focus();">
        <h2>Place Focus</h2>
        <form name="regform">
            <fieldset>
                <legend>Registration Form</legend>
                Name: <input type="text" name="fullname"
                    size="24" />  
                Email: <input type="text" name="email" /><br />
```

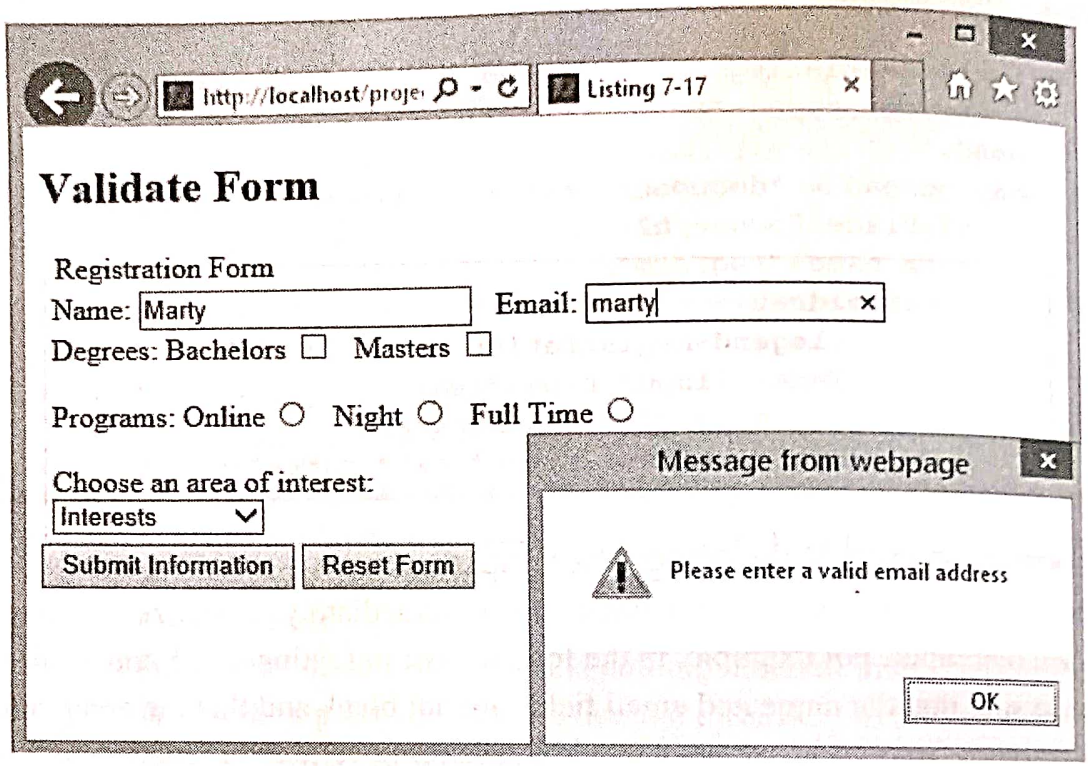
Recognizing and Validating User Input

Since JavaScript is running in the client, it can immediately recognize user input and validate that input. For example, in the form shown in Listings 7-15 and 7-16, we want to make sure that the name and email fields are not blank and that the email address has an @ symbol. To do this, we need to add a JavaScript validation function in the `<head>` `<script>` section and an `onsubmit` event in the `<form>` tag to recognize when the user is trying to leave the form and call the validation function. `onsubmit` is triggered by the HTML Submit button at the end of the form, which in turn can be triggered by pressing ENTER any place in the form.

There needs to be three conditional (`if`) statements in the JavaScript validation function, two to test if anything has been entered into the name and email fields, and one to test if the email entry has a @ in it. The @ test uses the JavaScript `indexOf()` method to look through a string for the existence of another string, the single letter "@" in our case, and returns the position where the second string starts. If the second string does not exist in the first string, `indexOf()` returns a "-1," so a test for <0 will tell if the "@" is missing. Listing 7-17 shows the changes that have been made to the code shown in Listings 7-15 and 7-16. The alert message that is displayed when an email address does not have an "@" is shown next.

CAUTION

The JavaScript `indexOf()` method is case sensitive. You must capitalize the "O" or it will not work.



Listing 7-17 Validate Form

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-17</title>
        <script type= "text/javascript">
        function validateform(){
            if (document.regform.fullname.value==""){
                window.alert("Please provide your full name");
                document.regform.fullname.focus();
                 return false;
            }
            if (document.regform.email.value==""){
                window.alert("Please enter your email address");
                document.regform.email.focus();
                return false;
            }
            if (document.regform.email.value.indexOf("@") < 0){
                window.alert("Please enter a valid email address");
```

```
            document.regform.email.focus();
            return false;
        }
        else {
            window.alert("Thank you for your registration");
            return true;
        }
    }
    </script>
</head>
<body onload = "document.regform.fullname.focus();">
    <h2>Validate Form</h2>
    <form name="regform" onsubmit="return validateform();" >
        <fieldset>
            <legend>Registration Form</legend>
            Name: <input type="text" name="fullname" size =
                "24"/>  
            Email: <input type="text" name="email" /><br />
            Degrees: Bachelors <input type="checkbox"
                name="bachelors"/>   Masters <input
                type="checkbox" name="masters"/><p></p>
            Programs: Online <input type="radio" name="online"/>
                  Night <input type="radio" name="night"/>
                  Full Time <input type="radio" name=
                "fulltime"/><p></p>
            Choose areas of interest:<br />
            <select name="Interests" >
            <option selected value="">Interests</option>
            <option value="socialwork">Social_Work</option>
            <option value="clinical">Clinical_Psych</option>
            <option value="education">Education</option>
            </select>
        </fieldset>
        <input type="submit" value="Submit Registration" />
        <input type="reset" value="Reset Form" />
    </form>
</body>
</html>
```

Accepting and Filing Form Data in PHP

The big difference between PHP and JavaScript is that JavaScript operates in the client and PHP operates in the server and allows you to accept data entered by the user and save it on the server. In this section, we'll use PHP to take the data a user enters into an HTML form, put it into an array, and then write it onto a disk. To do this, the form must have names

in all <input> fields and must have a Submit button. The form shown in Listing 7-17 meets both of these requirements, and it has been shown that data can be entered into this form and validated, making it a good candidate to demonstrate the collection and filing of form data with PHP.

Collecting Data in a Form

To collect the data that is entered into the form requires that Listing 7-17 be modified to identify the method to be used to transfer the data to PHP, and that optionally you identify where to send the data so PHP can use it.

The HTML <form> tag is used to specify both the method of transferring the data collected in the form and where to transfer it. The primary methods used to transfer form data are get and post. These are added to the <form> tag using one of the method attributes: method= "post" or method= "get". The major difference between the two is how the data is transferred. get transfers data by attaching it to the URL that opens the web page, like this:

```
http://localhost/project07/listing07-18.php?fullname=Marty+Matthews&
email=marty@anisp.com&bachelors=on&online=on&Interests=socialwork
```

The data is added after the actual URL with a ? separating the two. Spaces are marked by a + and successive fields are separated by &.

The post method transfers data in the HTTP header that is sent to the server with a web page request URL, but not in it. This provides a bit of security, since the data being transferred is not quite so obvious.

Identifying where to transfer data is done in the <form> tag with the action attribute. If there is no action attribute, then it is expected that the current web page will use the form data (see the next section); otherwise, an action attribute is added to the <form> tag with a web page name (such as *awebpage.php*), either by itself if it is in the same directory or folder as the current page, or with a path in front of the page name (for example, */parentfolder/folder/awebpage.php*). Here is an example of the <form> tag with both the method and action attributes:

```
<form name= "regform" method= "get" action= "awebpage.php">
```

Extracting Form Data from an Array

PHP provides a set of "superglobal" arrays ($_GET, $_POST, and $_REQUEST) that allow you to utilize form data that is transferred with the get and post <form> methods once the form has been transmitted with a Submit button. The $_GET array automatically collects form data from named fields that has been transmitted with the get method, the $_POST array does the same thing with the post method, and $_REQUEST array will collect data

from either the `get` method or the `post` method, as well `cookie` (see Chapter 6). `$_GET`, `$_POST`, and `$_REQUEST` are associative arrays built into and immediately usable anywhere in PHP. There is no need to otherwise define them. The indexes or keys for the arrays are the field names defined in the form. For example, in Listing 7-17, if we:

- Add `method="get"` to the **`<form>`** tag so it looks like this:
  `<form name="regform" onsubmit="return validateform();" method="get">`
- Add the PHP code with either `$_GET` or `$_REQUEST` above `<!DOCTYPE html>`, like this:

```
<?php
print_r($_GET);
?>
<!DOCTYPE html>
<html>
```

- Rename the script with .php.

Then, when the form is filled in and the Submit button is clicked, the array is displayed above the form, as you can see in Figure 7-1 where the keys are field names and the values are what was entered into the fields (the form is also normally reset—blanked out—but I filled it in again so you could see where the values in the array came from).

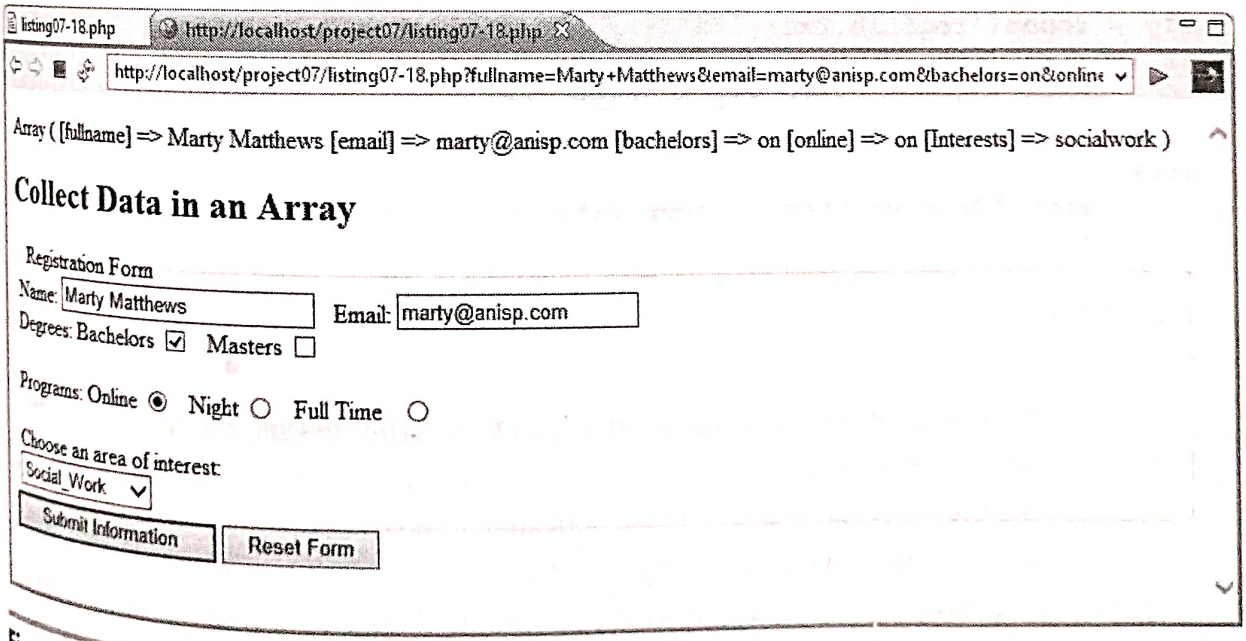


Figure 7-1 PHP's ability to immediately have available information entered into a form is one of its greatest strengths.

TIP

A common question is whether to use `$_REQUEST` in place of either `$_GET` or `$_POST`. There is no inherent difference. They work the same way and give you the same information. The principal reason to use `$_GET` or `$_POST` is that it clearly communicates to the script reader how the array got its information, and the reader can then look for the `get` or `post` that generated the data.

Writing Data to a File

While it is neat to display information entered into a form, you most likely want to save it in a file. This is done by:

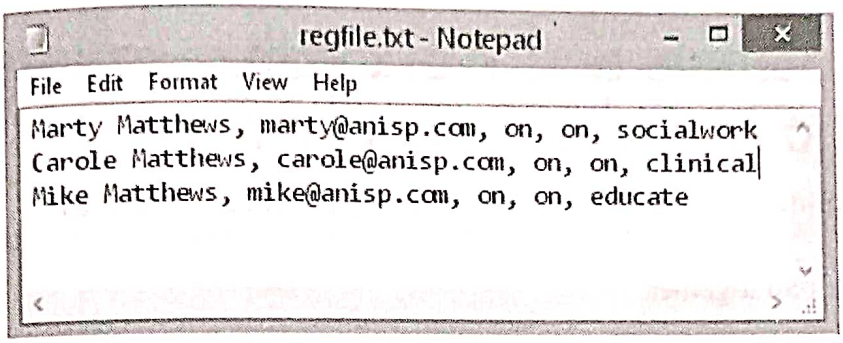
- Converting the `$_GET` array into a string
- Appending the `/r/n` carriage return and linefeed escape characters to the end of the new string
- Opening the file so you can write at its end
- Writing an entry or record onto the file and testing to see that was done

This is done by adding to the PHP script at the top of Listing 7-17 as shown here:

```
<?php
print_r($_GET);
$regrec = implode(", ", $_GET); //turn array into a string
$finalrec = $regrec . "\r\n" ; //add carriage return linefeed
$fp = fopen("regfile.txt", "a"); //open file to write at end
if (fwrite ($fp, $finalrec) == FALSE) {  //write the record
      echo "Cannot write fle.", "<br />";
      }
else {
      echo "File written.", "<br />";
      }
?>
<!DOCTYPE html>
<html>
```

See the discussion of the `implode()` array function earlier in this chapter, and review the file handling discussion in Chapter 6. There are several ways to join or concatenate two strings, but probably the simplest is the concatenation string operator, which is a simple period that is used in the preceding script to join a carriage return and linefeed to one person's registration record. The reason you want to do that is to provide delineation between records so you know where one record ends and another begins. Several PHP file functions depend on the linefeed character (`\n`) to identify the end of the record, while working with text files uses the carriage return for the same purpose.

You can see the result of writing a file in two ways. The simplest and least elegant is to locate and double-click the file. The result is shown next. The second is to write a small PHP script, shown in Listing 7-18a, that uses the `file()` function to read the named file into an array and then displays the array, shown following the listing.



Listing 7-18a Read Registration File

```
<!DOCTYPE html>
<html>
    <head>
        <title>Listing 7-18a</title>
    </head>
    <body>
        <h1>Read Registration File</h1>
        <?php
            $regarray = file("regfile.txt");
              print_r($regarray);
        ?>
    </body>
</html>
```

Chapter 8 will demonstrate additional PHP capabilities and how they can be used to create a user authentication system. Also, the work with MySQL in later chapters of this book will demonstrate a much more elegant file-handling capability.

Try This 7-1 Create and Work with a Multidimensional Array

Create a multidimensional array with the `array()` function and then:

- Display it all together
- Extract one item from the array
- Use a loop to display individual `key=>value` pairs
- Sort the array on the values, keeping the association with the keys
- Create a form to enter information into the array
- Use JavaScript to validate some of the information being collected in the form
- Write the array onto a disk file and prove it has been written there

Chapter 7 Self-Test

The following questions are intended to help reinforce your comprehension of the concepts covered in this chapter. The answers can be found in the accompanying online Appendix A, "Answers to the Self-Tests."

1. What are arrays?
2. What are the two principal parts of an array?
3. All arrays have only numeric keys. True or false?
4. Once an array has been created, it can't be changed. True or false?
5. What does the `print_r()` function do?
6. What does the `foreach()` function do?
7. What are three of the six ways that arrays can be sorted?

8. What is the principal construct used to navigate in arrays, and what are some of the ways it is used?
9. What are three ways that array functions allow you to transform arrays?
10. What are three ways that arrays can be compared?
11. What is a multidimensional array?
12. What are the two primary HTML tags used with forms?
13. What are two ways that JavaScript can be used with forms?
14. What are the two ways that information entered into a form can be transferred to a PHP script?
15. What are the PHP superglobal arrays used to collect form information?

Key Skills & Concepts

- Create an Index Script
- Add Registration Scripts
- Insert Sign-In Scripts
- Attach a Site Page

In this chapter we'll look at an example of how PHP can be put to use by itself in a user authentication system where we will set up a user login form and then validate the login information. This is not meant to be a system that you might use, but rather an example of how PHP can be used.

User Authentication

User authentication allows you to limit who has access to a website. It requires that all users register for the site, providing a name, a unique ID (usually an email address), and a password. This information is stored in a disk file with the ID and password encrypted so they cannot be misused. The fact that a person has registered (not their ID and password) is placed in a cookie on the user's computer, so he or she will not be asked to register again. A user wishing to enter a site is asked to enter his or her ID and password, which are encrypted and compared against the encrypted value in the file of registered users. If the ID and password correctly match a file entry, the user is admitted to the site. The fact that a user has been admitted to the site (not his or her ID and password) is stored in a session variable so the user can freely move from one site page to another without having to reenter the information.

Figure 8-1 shows a flowchart of a system to perform the user authentication function. It includes six scripts in four areas:

- **The initial entry** in index.php script, which checks if the user has a cookie
- **The registration area** with two scripts, one for the user to enter her or his information, and the other to encrypt the user information, write it to disk, and create and set a cookie

- **The sign-in area** with two scripts, one for the user to enter his or her information, and the other to encrypt the user information, compare it with the disk information, and create and set a session variable
- **A regular site page**, which checks for the presence of the session variable

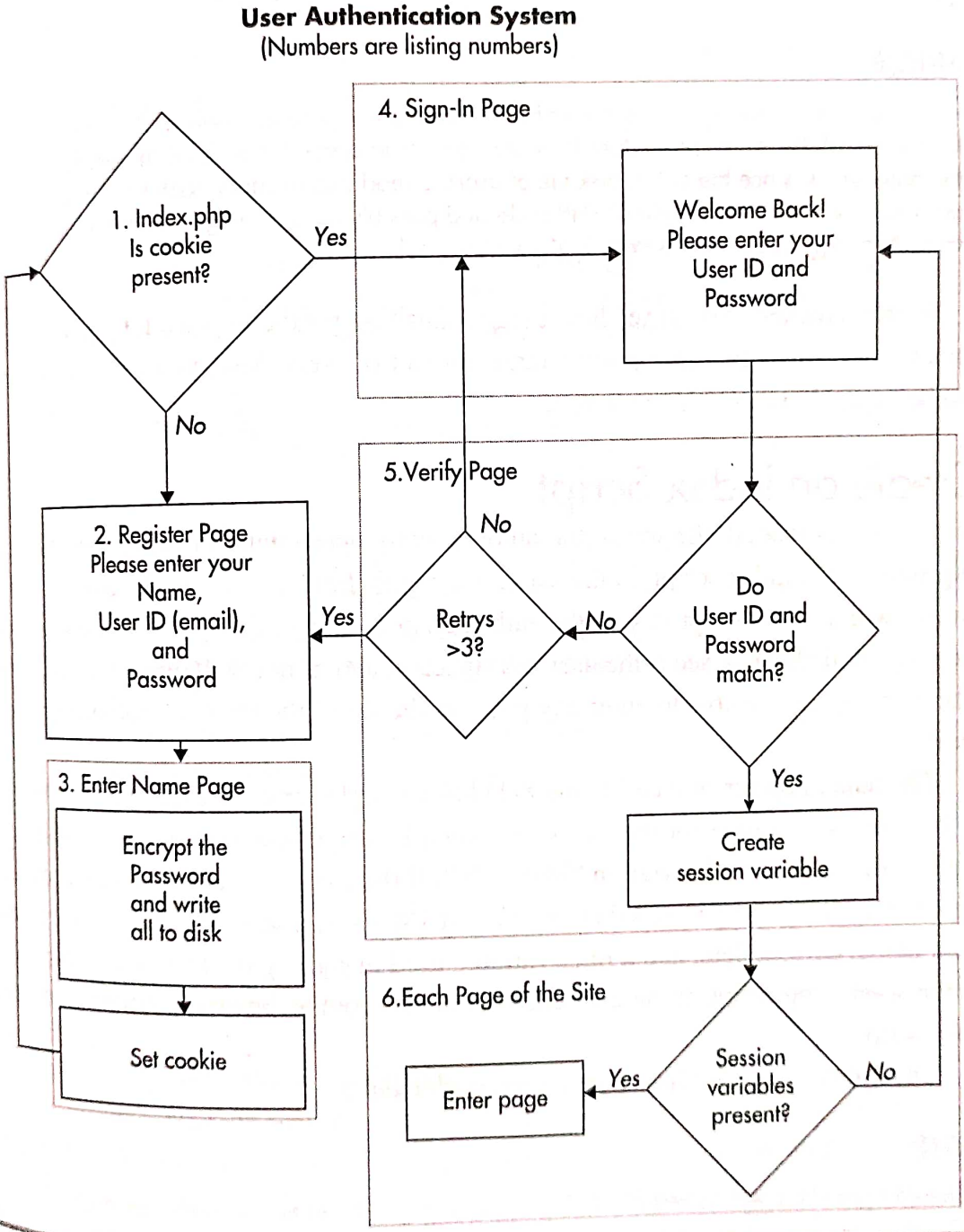


Figure 8-1 User authentication system flowchart

Each of these will be discussed in the following sections. I have tried to make these scripts as simple as possible to make the PHP process described here clear. In doing so, I have not included a CSS script to improve the formatting and make the web pages more attractive. As a result, I have used several deprecated HTML formatting attributes such as `width`, `cellpadding`, and `cellspacing` that HTML5 considers obsolete and should be replaced with CSS. I fully agree that CSS is the best way to do the formatting, but my objective here is to focus on PHP and how it can be used by itself.

CAUTION

This user authentication system is meant for demonstration purposes *only* and is *not* intended as a fully secure procedure to protect access to a site. It also is not meant for very many users, since the entire disk file of users is read into memory each time it is used. Use it only as an example of PHP code and possibly as a starting point for your own system. *Any use of this system is at your own risk.*

In later chapters, you'll see how using a database for the files would greatly help a user authentication system and support a large number of users. Also, you will see how a CSS can be applied to a system such as this.

Create an Index Script

When a site is loaded, the script that automatically opens without being specifically requested is the index script, in this case, index.php. For that reason, the user authentication system will use that script to start the authentication. If any other script in the site is opened, the script will check to see if the user has signed in and, if not, will direct the user to sign in. The user will not be able to enter any page on the site if the page has the protection code on it.

The index.php script (see Listing 8-1) has a simple series of PHP statements that asks if the user has a cookie for the site. (The example scripts use a site name of Matthews Technology and the abbreviation "MatTech.") If the user has a cookie from a previous visit to the site and registered there in the last 180 days, a session variable is created with their name, current date, and a retry variable used in signing in. The user is then directed to the signin.php script. If the user does not have a cookie, he or she is sent to the register .php script.

The index.php script has several unique elements:

NOTE

It should again be emphasized that this is a simple example. In normal websites, the index.php should contain the home page information, not only for the reader, but for bookmarking and search engine purposes.

- The script is pure PHP code, has no HTML code, and nothing is displayed to the user from that script. You'll see we do that several times in this system.
- The script uses the `isset()` function to determine if the cookie exists. Remember that if a domain has set a cookie on a user's computer, it will be automatically returned to the domain's server the next time the user connects to the server as long as the user does not clear their cookies.
- The `header("Location:... ")` function is used to transfer execution to another web script by using the HTML header to inform the server to send another script. This function *must* be executed before any other output sent to the user, such as an `echo` statement or the `<html>` or `<head>` elements.

TIP

Start out with a lot of `echo` statements to assist in the debugging, and use HTML `<a href=...>` (anchor) elements in place of the PHP header statements, since `header` cannot be used after any output, for example, the `echo` statements.

Listing 8-1 index.php

```
<?php
   session_start();
   //Check if user has a cookie.
   if (isset ($_COOKIE["MatTech"])) {
         //If so, set session and go to sign in.
         $_SESSION["name"] = $_COOKIE["MatTech"][name];
         $_SESSION["retry"] = 0;
         $_SESSION["time"] = time();
         header( "Location: signin.php");
   }
   else {
         //If not, go to registration.
         header( "Location: register.php");
      }
?>
```

Add Registration Scripts

As mentioned, there are two registration scripts: register.php, which handles the user input, and enterName.php, which does the processing. It is possible to do all this in one script, as you will see in some of the PHP/MySQL examples later in this book, but the two-script solution provides a clean introduction to this process.

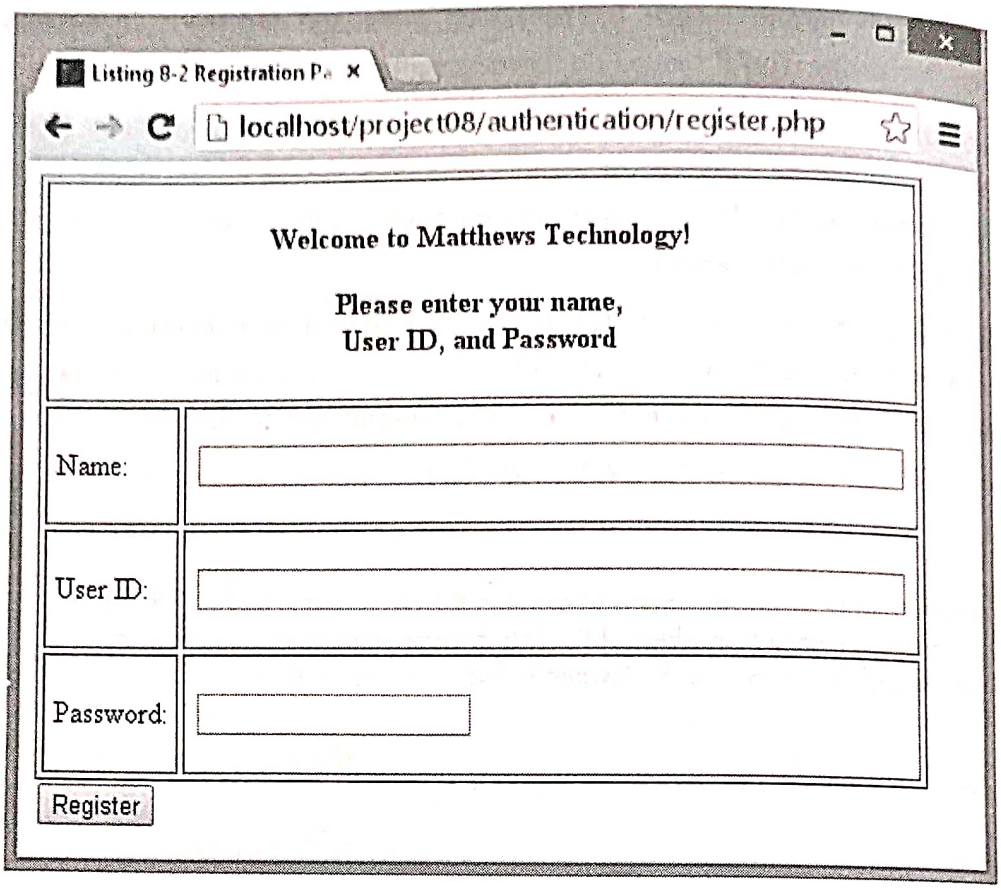


Figure 8-2 The registration form could use a CSS to spruce it up a bit.

Input Form

The register.php script, which you can see in Listing 8-2, is a table containing a form with three fields, Name, User ID, and Password, as shown in Figure 8-2. When the user completes filling out the form and clicks Register, the form `action` transfers execution to the enterName.php script and populates the `$_POST` superglobal array with the contents of the form.

TIP

JavaScript can be used with `document.body.onload` event to automatically go to the first form field, but it is not included here to focus on PHP.

Listing 8-2 register.php

```
<!--User enters name, ID, and password.-->
<html>
  <head>
    <title>Listing 8-2 Registration Page</title>
  </head>
  <body>
    <div id="form">
      <!-- Go to enterName.php after clicking Register -->
      <form action="enterName.php" method="post" id="registerForm">
      <table width="150" border="1" cellspacing="3" cellpadding="5" >
        <tr height= 50>
          <th colspan= "2" valign="middle" >
            <p id="head">Welcome to Matthews Technology!</p>
            <p id="body">Please enter your Name,<br />User ID, and
              Password</p>
          </th>
        </tr>
        <tr>
          <td width="40">
            <p class="label">Name:</p>
          </td>
          <td width="100">
            <input type="text" name="name" value="" size="60" />
          </td>
        </tr>
        <tr>
          <td>
            <p class="label">User ID:</p>
          </td>
          <td>
            <input type="text" name="userid" value="" size="60" />
          </td>
        </tr>
        <tr>
          <td>
            <p class="label">Password:</p>
          </td>
          <td>
            <input type="password" name="passwd" value="" size="20" />
          </td>
        </tr>
      </table>
        <input type="submit" name="submit" value="Register" />
      </form>
    </div>
  </body>
</html>
```

Encryption and Save to Disk

The enterName.php script, shown in Listing 8-3, is the second pure PHP script. It picks up the name entered at registration and passed to this script through the `$_POST` superglobal array. This is combined with the current time and the expiration time to create and set a cookie on the user's computer. The user ID and password, entered at registration, are combined and then encrypted with SHA1 (Secure Hash Algorithm 1) for 160-bit encryption. Next, the current time, the user's name, and the encrypted ID and password are placed into an array, and the array is written to disk with the `file_put_contents()` function because it allows an array to be appended to an existing disk file. System execution is then transferred back to index.php.

Listing 8-3 enterName.php

```
<?php
/* Cookie is written. Then the user id
*  and password are encrypted and that and
*  the name are written to disk. */

        //Information for cookie, expires in 6 months.
        $name = $_POST['name'];
        $date = time();
        $expire = time()+(60*60*24*180);

        //Set cookie.
        setcookie("MatTech[name]", $name, $expire, "/");
        setcookie("MatTech[date]", $date, $expire, "/");

        //Combine userid and password and encrypt it.
        $userPasswd = $_POST['userid'] .= $_POST['passwd'];
        $encryptid_pw = sha1($userPasswd);

        //Build an array of data and write it to disk.
        $entry = array( 'index' => time(),
                        'name' => $name,
                        'encrypt' => $encryptid_pw
                      );
        if (!$byteswrite = file_put_contents('namelist.txt',
           $entry, FILE_APPEND)) {
            echo "<br />File not written.<br />";
           }

        //Return to Index.php.
        header( "Location: index.php");
?>
```

Insert Sign-In Scripts

Once again, the two sign-in scripts, signin.php and verify.php, could be in one script, but are split for the clarity of this exercise.

Signing In

Once the user has a cookie on her or his computer, the user goes immediately to signin.php. This is again a form within a table, shown in Figure 8-3 and Listing 8-4. When the user clicks Sign In, the form `action` transfers the execution to verify.php and the `$_POST` superglobal will be populated with the user ID and password that the user entered.

NOTE

You should not get to signin.php without being registered, but the "Not registered? Click here!" is added so you can get back to register.php without having to delete the cookie.

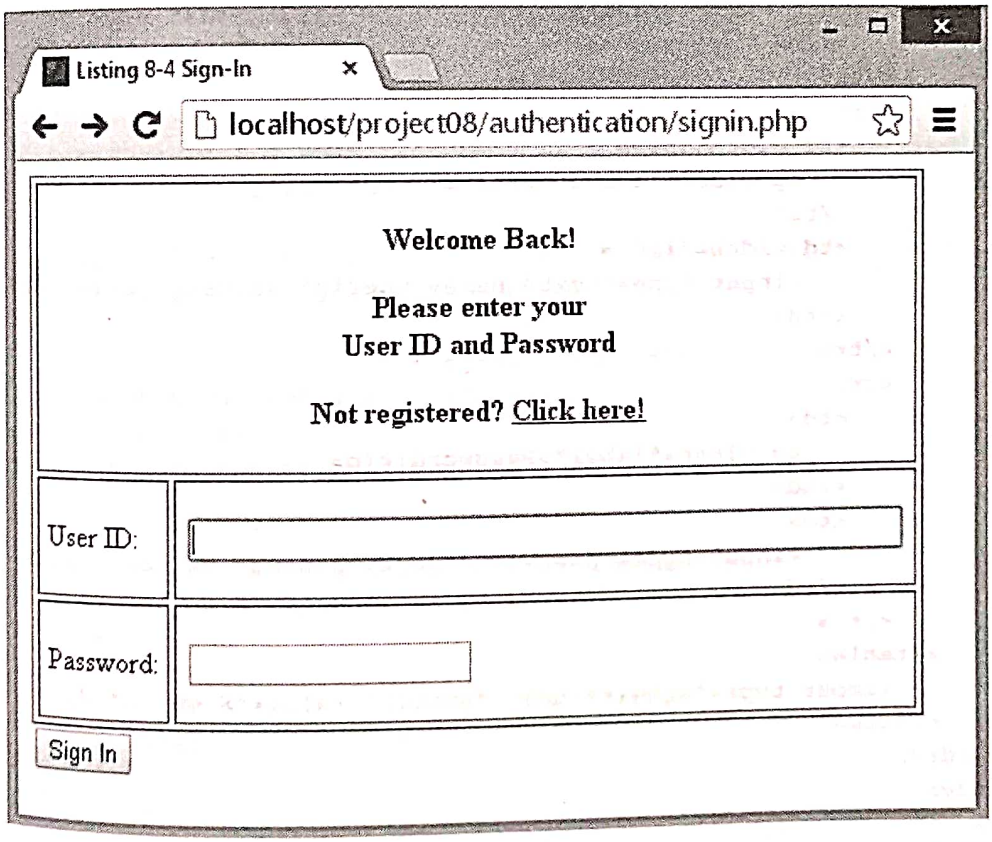


Figure 8-3 The fact that the user ID and password are transmitted over the Internet unencrypted is a major security flaw. Using HTTPS to do this would help.

Listing 8-4 signin.php

```
<!-- User enters id and password. -->
<html>
   <head>
      <title>Listing 8-4 Sign-In</title>
   </head>
   <body>
      <div id="form">
         <!-- Display the sign-in form. After filling in, go to
              verify script. -->
         <form action="verify.php" method="post" id="signinForm">
         <table width="100" border="1" cellspacing="3" cellpadding="5" >
            <tr height="50">
               <th colspan="2" valign="middle" >
                  <p id="head">Welcome Back!</p>
                  <p id="body">Please enter your<br />User ID
                      and Password</p>
                  <p id="body">Not registered? <a href="register.php">Click
                     here!</a></p>
               </th>
            </tr>
            <tr>
               <td width="50">
                  <p class="label">User ID:</p>
               </td>
               <td width="120">
                  <input type="text" name="userid" value="" size="60" />
               </td>
            </tr>
            <tr>
               <td>
                  <p class="label">Password:</p>
               </td>
               <td>
                  <input type="password" name="passwd" value="" size="20" />
               </td>
            </tr>
         </table>
            <input type="submit" name="submit" value="Sign In" />
         </form>
      </div>
   </body>
</html>
```

Verifying the Input

The verify.php script, which you can see in Listing 8-5, reads the entire namelist.txt file into the `$namelist` string using the `file_get_contents()` function. It then combines

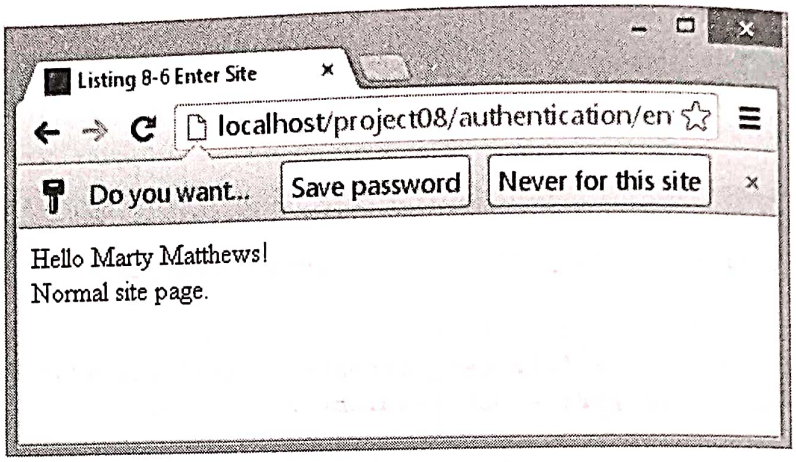
the user ID and password that were entered at sign-in and encrypts that combination. The `strpos()` function is then used to see if the encrypted combination is contained in `$namelist`. If the encryption is there, the session variable is updated, and execution is transferred to the original site scripts. If the encryption is not found on the disk file, the `$retry` variable is incremented and checked to see if it is greater than 3. If so, execution is transferred to register.php; otherwise, the `$retry` variable execution is stored in the `$_SESSION` variable, and execution is transferred to signin.php.

Listing 8-5 verify.php

```
<?php
   session_start();

//  Verify user's id and password, and create session.

      //Read the name list file.
      if(!$namelist = file_get_contents('namelist.txt')) {
         echo "<br />File not read.<br />";
      }

      //Combine the user ID and password and encrypt it.
      $userPasswd = $_POST['userid'] .= $_POST['passwd'];
      $testentry = sha1($userPasswd);

      //Determine if the encrypted user ID and password are in file.
      if(strpos($namelist, $testentry) >= 0){

         //If there, reset the session and enter site.
         $_SESSION["retry"] = "admit";
         $_SESSION["time"] = time();
         header( "Location: enterSite.php");
      }
      else {
         //If not, add to Session Retry and test > 3
         $retry = $_SESSION["retry"];
         $retry++;
         if ($retry > 3) {
            //If greater than 3 go to register.
            header( "Location: register.php");
         }
         else {
            //If less than 3, reset Session Retry and go to Sign in
            $_SESSION["retry"] = $retry;
            header( "Location: signin.php");
         }
      }
?>
```

Attach a Site Page

The code snippet that goes on each regular site page (see Listing 8-6) checks to see if `$_SESSION["retry"]` is present and equal to "admit." If so, the user is greeted, and execution flows into the regular page, as shown next. Otherwise, execution is returned to index.php, where the user has to prove she has a cookie, or she will have to register, and in either case, she will have to sign in.



TIP

Instead of including the snippet of PHP code in Listing 8-6 on each regular site page, save just the PHP snippet as its own file; then as the first line of every site page, insert `<?php require_once(enterSite.php); ?>`. One feature of the `require` and `require_once` functions is that the balance of the script on the page will not execute until the code referenced by the function successfully executes. It halts execution with a fatal error. This contrasts with the `include` and `include_once` functions, which do not halt execution of the remaining script. The `_once` prevents the snippet from being loaded more than once.

Listing 8-6 enterSite.php

```
<?php
   session_start();
//Having successfully signed in...
   //Check to see if the session variable is present and is "admit"
   if (isset($_SESSION["retry"]) && $_SESSION["retry"] == "admit") {
      //If so, continue.
      echo "Hello ", $_SESSION["name"], "!<br />";
   }
   else {
      //If not, return to the site Index page.
      header( "Location: index.php");
   }
?>
```

Many things can be done to enhance this system. The first is probably to make the pieces that collect the user ID and password use HTTPS and thus secure that transfer. The second is to use a CSS to make the data collection forms look better and not use deprecated code. Two other areas that we'll address later in this book are better handling of error messages, such as those returned from disk read and write errors, and handling the user trying to enter unusual characters and other content (like pasting an image) into the form.

Try This 8-1 Create a PHP System of Scripts

Using the example in this chapter, create your own system of scripts that:

- Create and reference a cookie to carry information from session to session and web page to web page within one session.
- Create a form in which a user can enter information.
- Encrypt some of the user information and save it and other user information to disk.
- In a separate session, verify the user information before letting them enter the site.
- In this system, pass user-entered information from a form to the script and from one script to another.

Chapter 8 Self-Test

The following questions are intended to help reinforce your comprehension of the concepts covered in this chapter. The answers can be found in the accompanying online Appendix A, "Answers to the Self-Tests."

1. What function is used to test if something exists?
2. What function is used to transfer to another script and what are its constraints?
3. What is used to transfer data entered into a form to the script itself and what is it called?
4. What should I have used in place of `width`, `cellspacing`, and `cellpadding` attributes?
5. How does one encrypt a password?
6. What is the technique and function used in this example to test if a password is valid?
7. How do you refer to an external piece of PHP code and what are some of its options?

Key Skills & Concepts

- Understanding Databases
- Understanding a Relational Database
- Get and Install phpMyAdmin
- Opening and Exploring phpMyAdmin
- Create and Use a Database in phpMyAdmin

With PHP alone, you saw how to use arrays to collect, store, and retrieve small amounts of information and save that information on disk. While the PHP tools work, you can readily see that they would not work well with large volumes of information. For that task, you need a more robust tool, a "database" able to securely handle such volume and provide the means to quickly search for and retrieve information in a number of different ways. In this chapter, we'll look at what a database is, the different kinds of databases, the parts of a database, how a database is used, and how to use phpMyAdmin to set up and maintain a database for use on the Web.

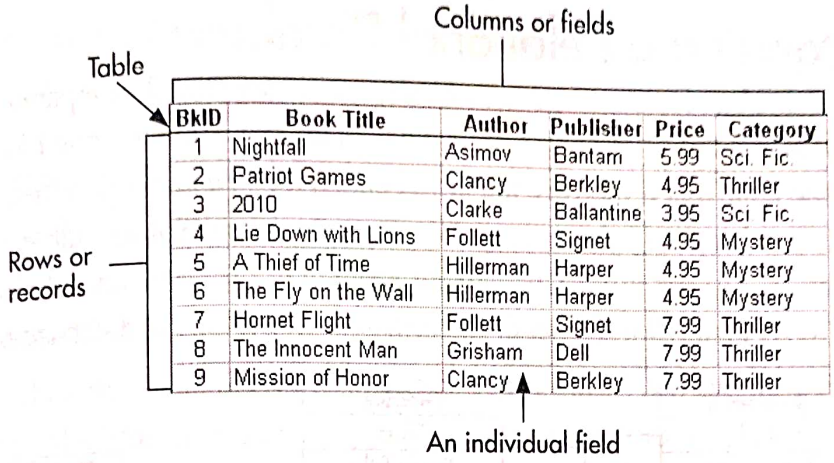
Databases and Relational Databases

A *database* is an organized way of storing information. The key is the word "organized." The primary purpose of a database is to store information in such a way that it can be easily and quickly found and retrieved. A database must facilitate the easy and quick retrieval and use of the data it has collected.

Understanding Databases

In its simplest form, a database is a *table* of information, as shown in the Excel list of books in Figure 9-1. Each book entry is a *row* in the table and is called a *record*. Each piece of information in a book entry, such as the title or author, is a *column* in the table, and is called a *field*.

The table in Figure 9-1 is called a *flatfile*, where all the information in the database is in a single table. Notice that there are several books that have the same author, the same publisher, or the same category. If these fields were made into their own tables



BkID	Book Title	Author	Publisher	Price	Category
1	Nightfall	Asimov	Bantam	5.99	Sci. Fic.
2	Patriot Games	Clancy	Berkley	4.95	Thriller
3	2010	Clarke	Ballantine	3.95	Sci. Fic.
4	Lie Down with Lions	Follett	Signet	4.95	Mystery
5	A Thief of Time	Hillerman	Harper	4.95	Mystery
6	The Fly on the Wall	Hillerman	Harper	4.95	Mystery
7	Hornet Flight	Follett	Signet	7.99	Thriller
8	The Innocent Man	Grisham	Dell	7.99	Thriller
9	Mission of Honor	Clancy	Berkley	7.99	Thriller

Figure 9-1 A database is a table with records and fields.

that can contain additional information, it would simplify the Books table, provide more information, save storage space, and make it easier to change, as you would only have to update in one place. (Savings are very small in this database, but if you have a large database, they would be significant.) The resultant four tables, which are shown in Figure 9-2, form a *relational database.*

BkID	Book Title	Aid	Pid	Price	Cid
3	2010	3	1	3.95	2
5	A Thief of Time	6	5	4.95	1
7	Hornet Flight	4	6	7.99	3
4	Lie Down with Lions	4	6	4.95	1
9	Mission of Honor	2	3	7.99	3
1	Nightfall	1	1	5.99	2
2	Patriot Games	2	3	4.95	3
6	The Fly on the Wall	6	5	4.95	1
8	The Innocent Man	5	4	7.99	3

Aid	Last Name	First Name
1	Asimov	Isaac
2	Clancy	Tom
3	Clarke	Arthur
4	Follett	Ken
5	Grisham	John
6	Hillerman	Tony

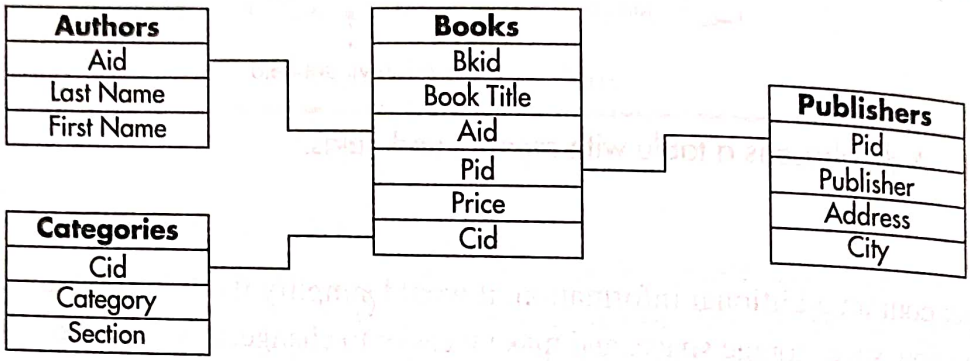
Cid	Category	Section
1	Mystry	E-9
2	Sci. Fic.	F-14
3	Thriller	G-10

Pid	Publisher	Address	City
1	Ballantine	201 E 50th	New York
2	Bantam	666 5th Ave	New York
3	Berkley	200 Madison A	New York
4	Dell	666 5th Ave	New York
5	Harper	10 E 53rd	New York
6	Signet	375 Hudson	New York

Figure 9-2 A relational database places repetitive information into separate tables.

Understanding a Relational Database

In a relational database, repeated pieces of information are stored in separate tables and related to one another through their *key*, or index, a unique value assigned to each record in the table. In the table that it indexes, the key is called the *primary key*. When that key is used in another table, it is called a *foreign key*. For example, in the Authors table, the Aid (author ID) is the primary key, but in the Books table, the Aid is a foreign key relating the Authors table to the Books table. The full set of relationships in the Books database is shown next.



The process of taking a field with repeated values and placing the field in its own table is called *normalization*. It not only reduces the space taken up by a database, but it also significantly enhances the maintainability of a database. Without normalization, if you wanted to change a value in a field, you would have to change all occurrences of the value everywhere in the database. With normalization, you only have to change a single occurrence.

Using phpMyAdmin

phpMyAdmin is an online graphical user interface for working with a MySQL database and is written in PHP. It is an app that can be added into Zend Server. The principal use of phpMyAdmin is to initially create databases and their tables that will be exercised programmatically with PHP and to do occasional maintenance on these databases.

In this section, we'll look at how to get and install phpMyAdmin, explore its interface, review its security and how to change it, create a database with a table, and look at the details of a database specification, including the various data types.

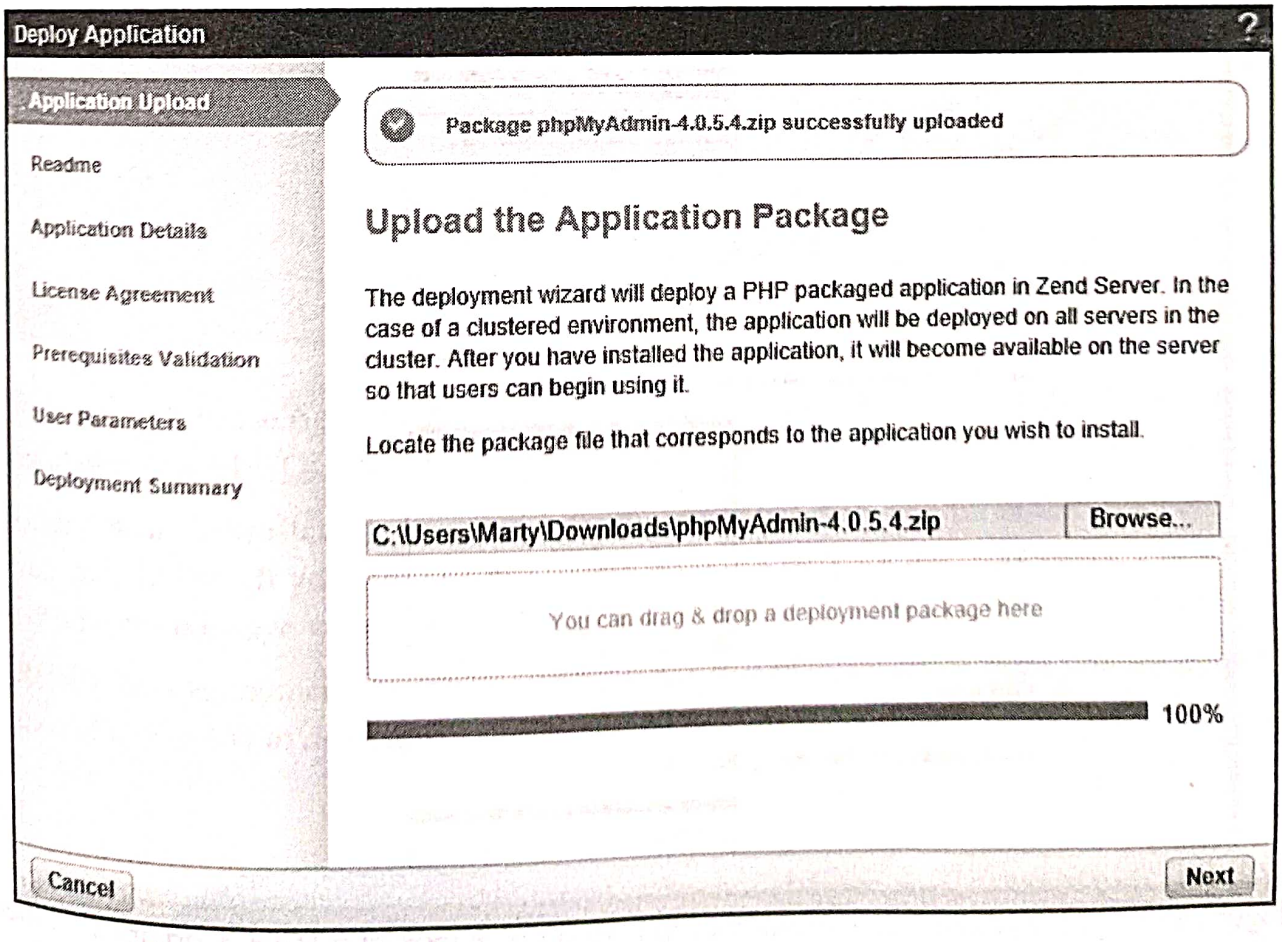
Get and Install phpMyAdmin

With the Zend Server, you will need to separately download and then install phpMyAdmin. Zend provides both instructions and links to do that here:

http://files.zend.com/help/Zend-Server-6/content/installing_phpmyadmin.htm

Open this website, shown in Figure 9-3, and use it and the following steps to install phpMyAdmin on your computer:

1. In the Zend Server phpMyAdmin help page, click the link ("here") in the first step to begin the download process.
2. Click Save As to choose where to save phpMyAdmin. I suggest you save the app in the recommended default, the user's Download folder. Click Save.
3. Double-click the Zend Server icon on your desktop, if you placed one there, or locate the app on the Start menu in Windows 7 and earlier versions or in the Apps screen of Windows 8.
4. Enter your Zend username (usually "admin") and password and click Login. This opens the user interface (UI) discussed in the phpMyAdmin Help page.
5. Click Applications | Apps in the toolbars at the top of the window, and then click Deploy Application.
6. Click Browse (or Choose Files), select the folder where you placed the file, and then select the file and click Open.



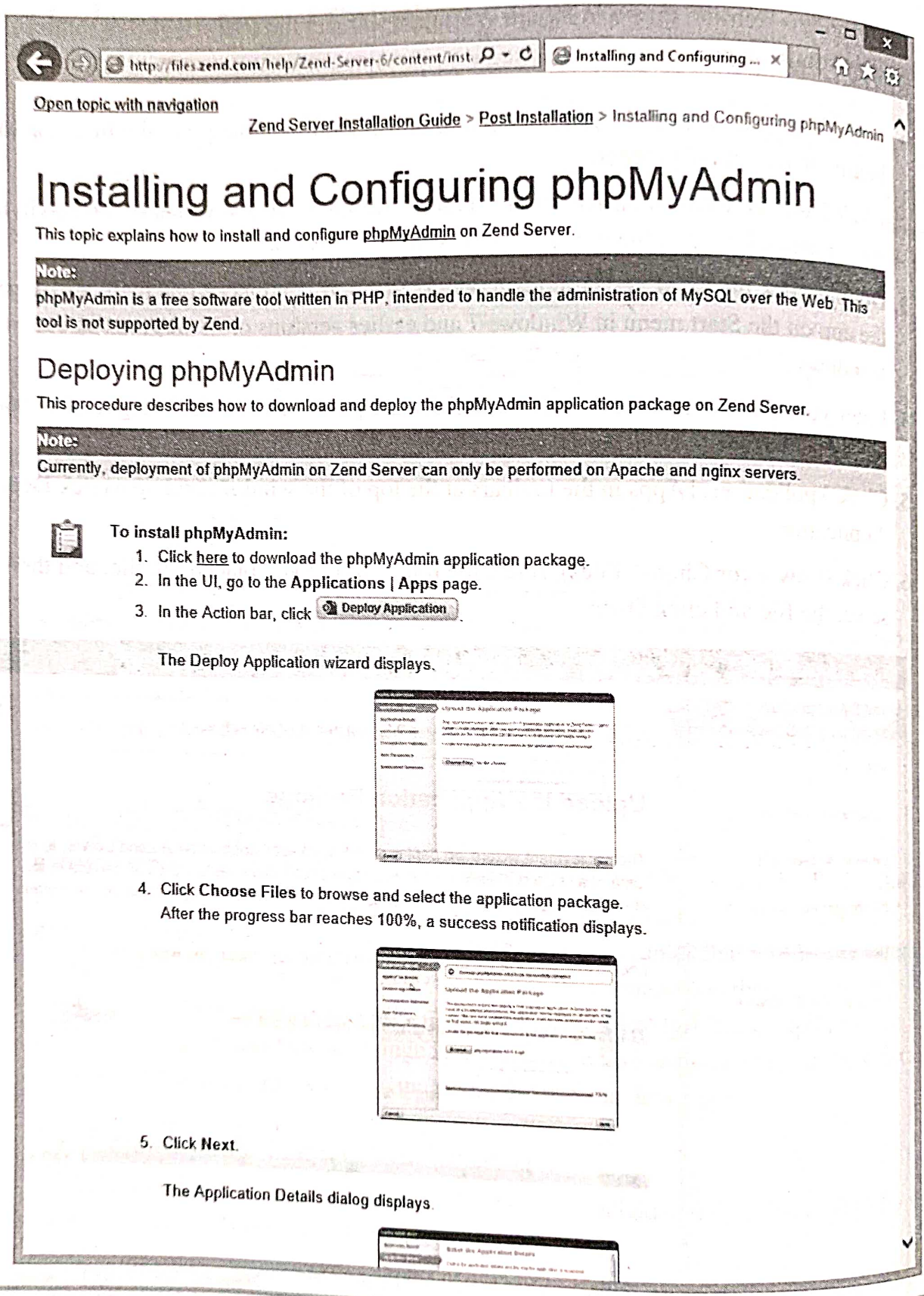
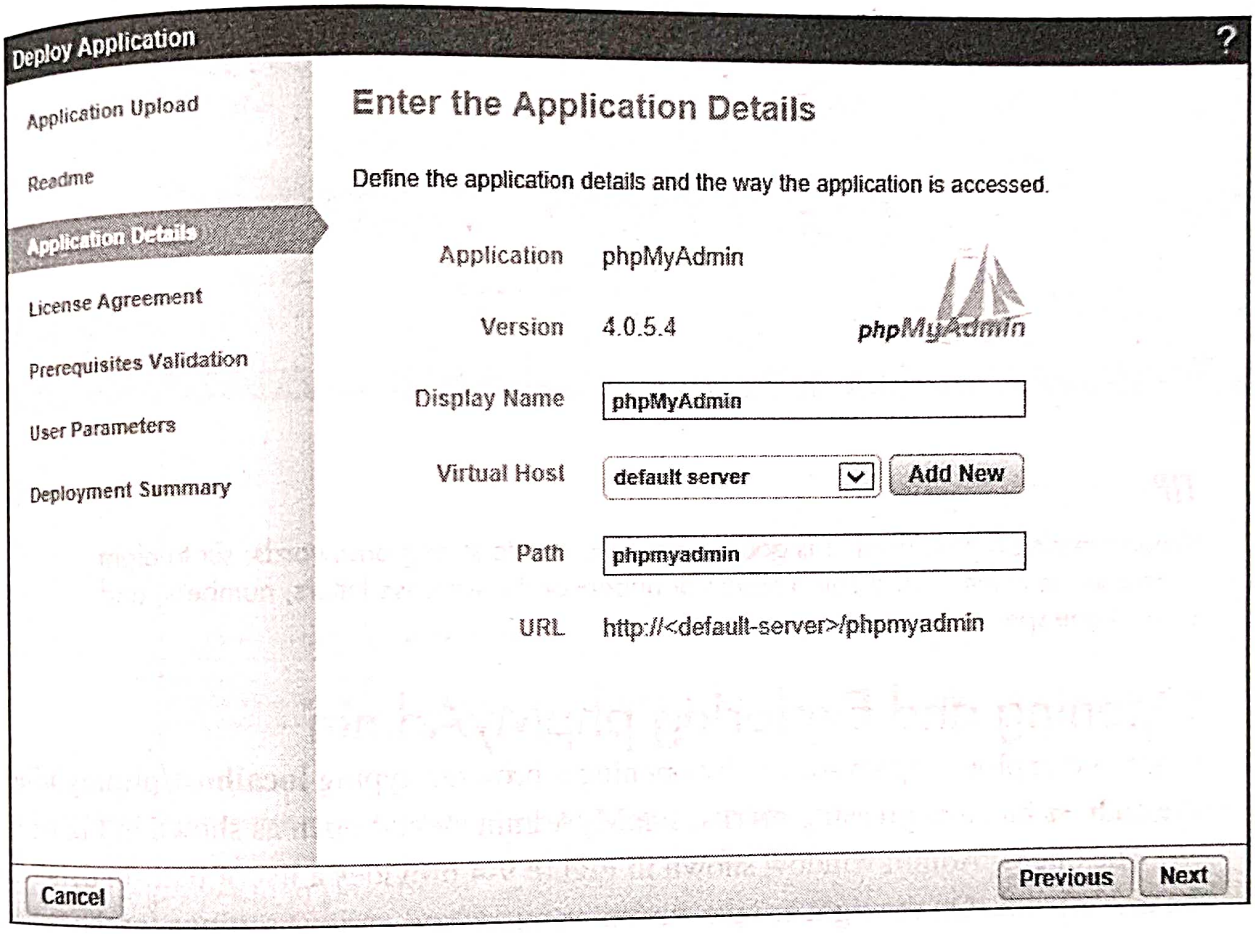


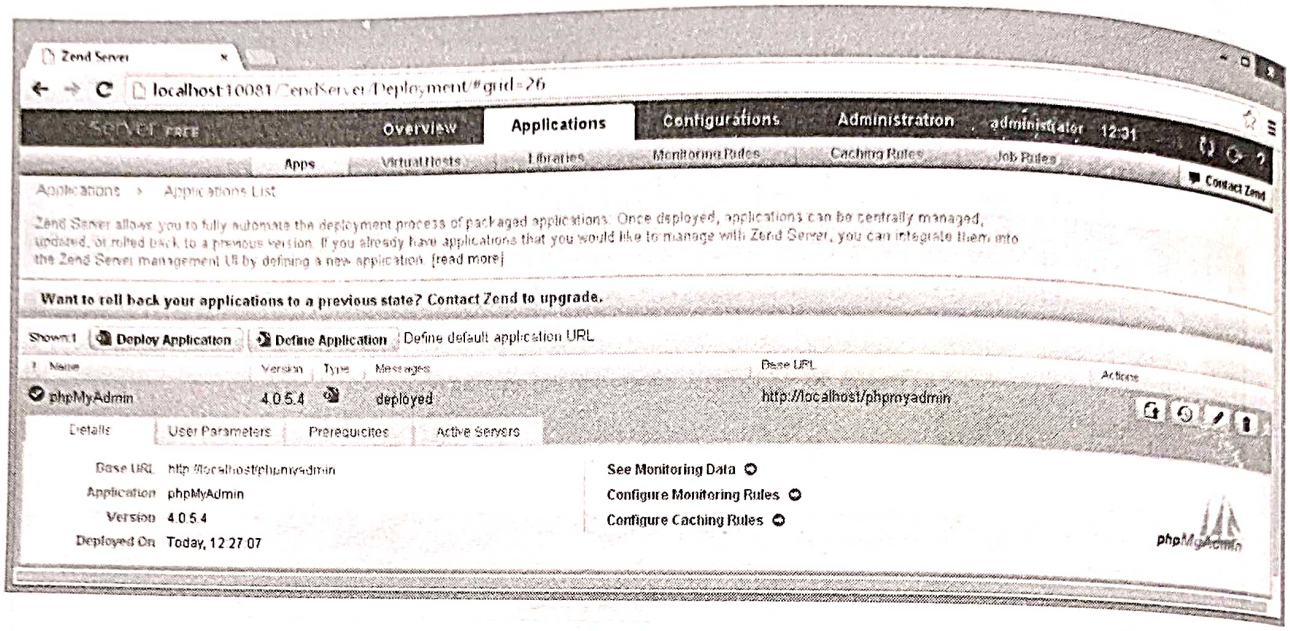
Figure 9-3 phpMyAdmin is a useful tool to take a quick look at a MySQL database without having to write a program to do it.

7. When you are told it has been successfully uploaded, click Next. Review the Readme page and again click Next.
8. Keep the suggested Default Name and Virtual Host (the default server will normally be "localhost"). For the Path, I would enter **phpmyadmin**, but it can be any folder you want. Click Next.



9. Review the License Agreement, click I Have Read, and click Next. You should see the validation that MySQL has been loaded. Click Next.
10. Under most circumstances, keep the default User Parameters and fill in the password you want to use. (If you don't enter a password, especially for the "root" user, you will get an error message when you open phpMyAdmin later in this chapter.) Click Next.
11. Review the Deployment Summary, use Previous to make any corrections you want, and then click Deploy to do that.

12. When the process is complete, click phpMyAdmin to display its details.



TIP

When creating a password, it is good practice to create strong passwords: six to eight characters in length and a combination of upper- and lowercase letters, numbers, and at least one special character.

Opening and Exploring phpMyAdmin

Open and explore phpMyAdmin by opening a browser, typing **localhost/phpmyadmin** in the address bar, and pressing ENTER. phpMyAdmin should open as shown in Figure 9-4.

The phpMyAdmin window shown in Figure 9-4 provides a list of built-in databases on the left side. On the right are the current versions and various settings for the database server (MySQL), the web server (Apache), and phpMyAdmin, as well as documentation for phpMyAdmin. In the middle are general and appearance settings. Across the top are ten tabs that provide a lot of additional information about the resident MySQL databases and the ability to change settings and, in the SQL tab, to do everything you can do in the MySQL command-line client.

If you didn't enter a password, you'll see a warning at the bottom of the window that phpMyAdmin's default settings do not include a password for the default username "root." This means that anybody who knows the username "root," and many people do, can access your database and do what they wish with it. While you are working on your own computer

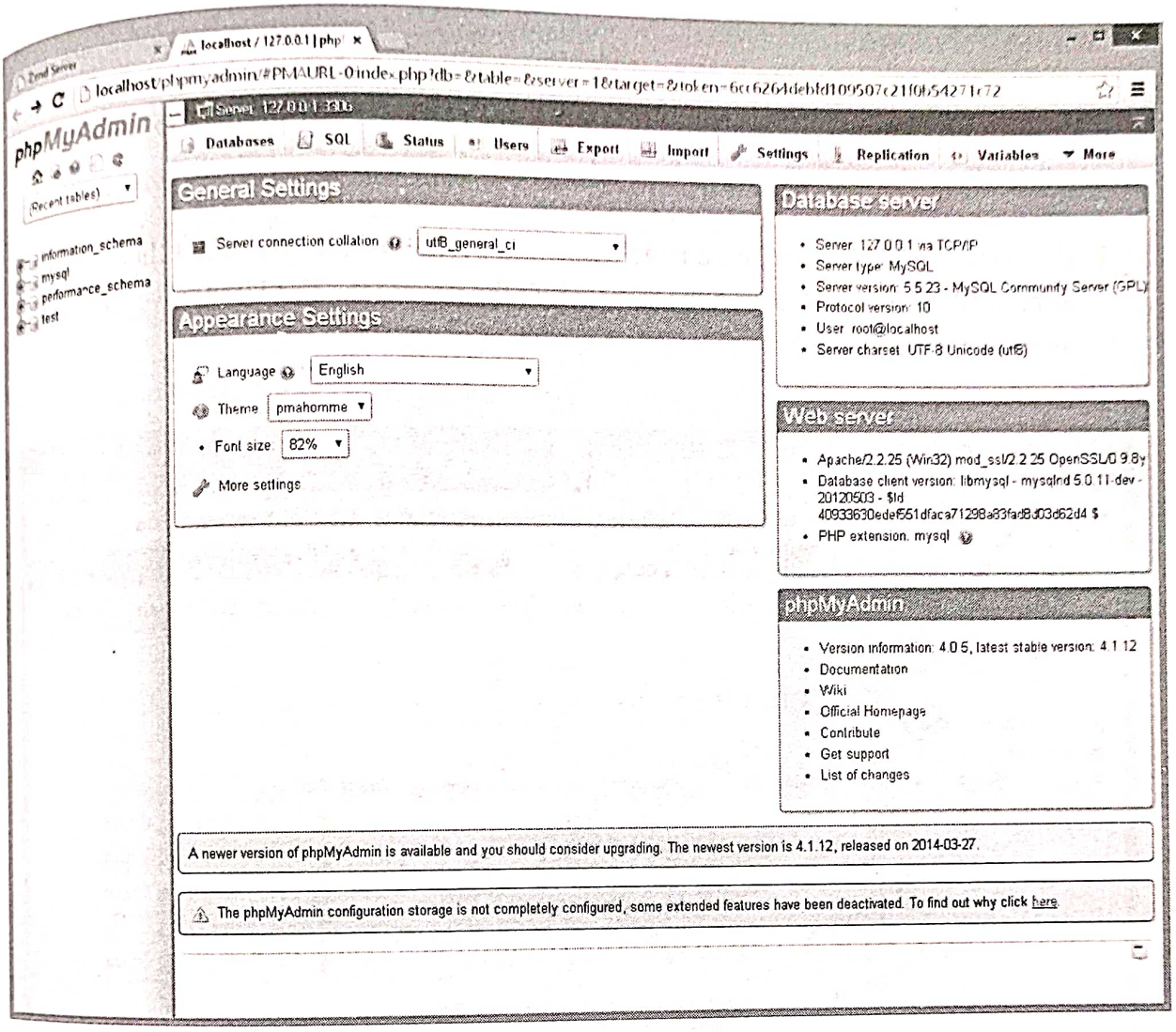


Figure 9-4 phpMyAdmin provides a control panel for setting up and maintaining MySQL.

using localhost (or 127.0.0.1), your risk is not great, but when you upload your site to a commercial host, it becomes critical that you set the username to something other than "root" and use a strong password.

NOTE

If you did not set a password for the "root" user while installing phpMyAdmin, you can use the Update feature in the Zend Server Application Apps window, opposite phpMyAdmin, to do that.

Setting Privileges

You can set and change privileges and add and delete users through the Users tab of phpMyAdmin—providing you have the privileges to do that, which the root user does (another reason to set a password for root, even on your own computer).

1. From the phpMyAdmin home page (shown in Figure 9-4; if you're not there, click Server:127.0.0.1:3306 above the tabs), click the Users tab. The Users Overview opens, as shown in Figure 9-5.

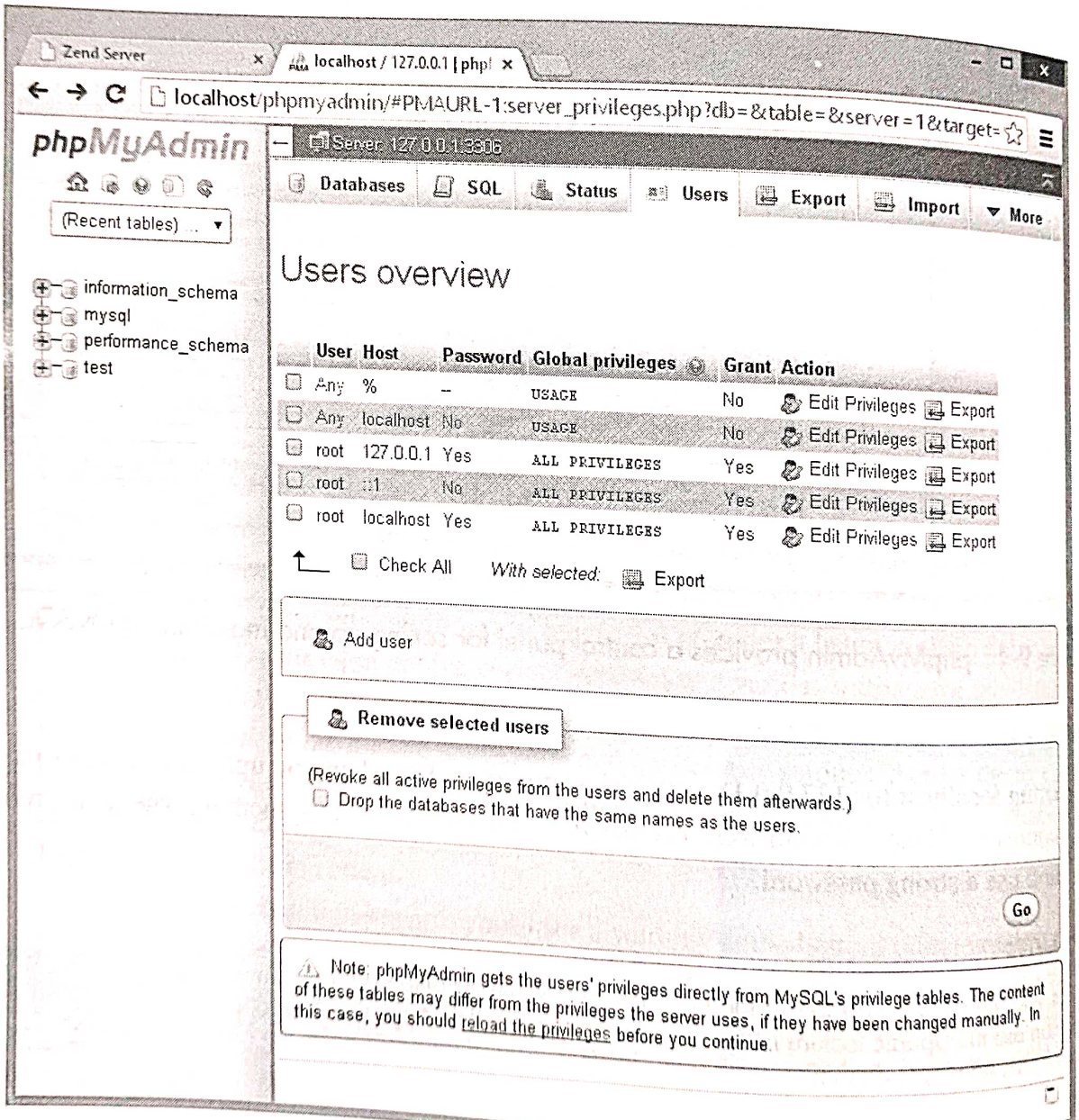
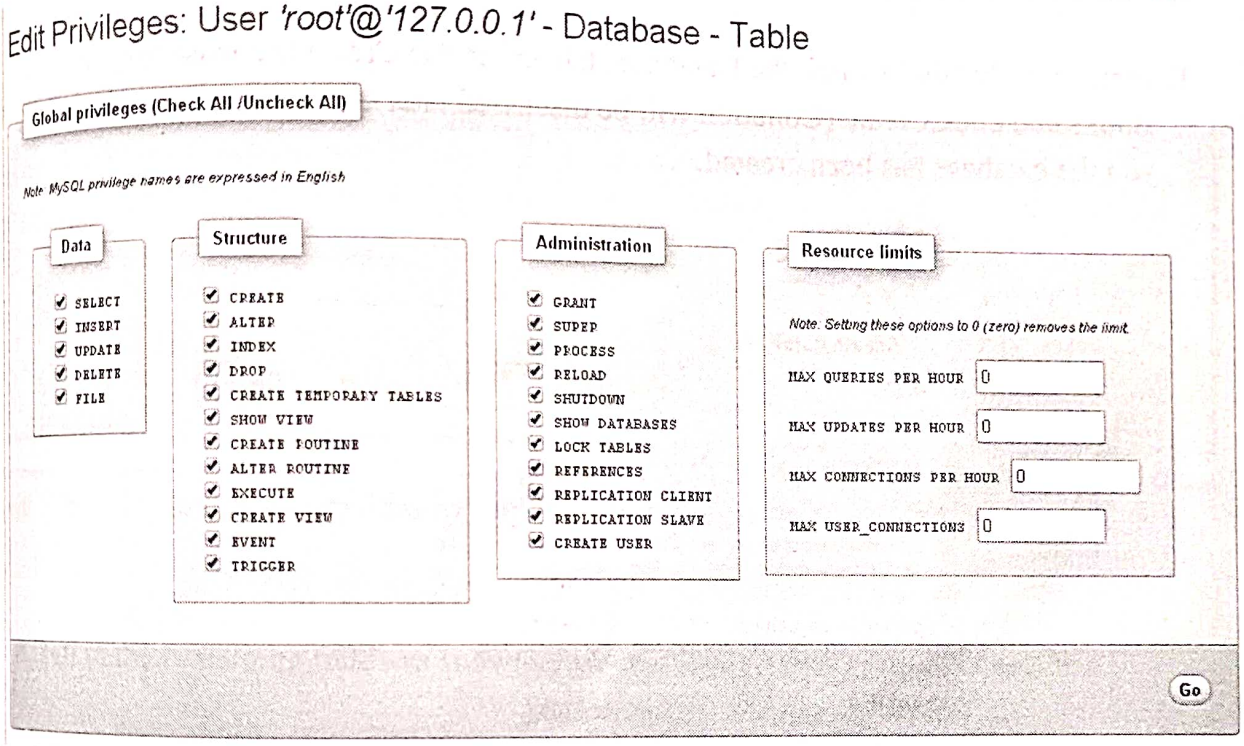


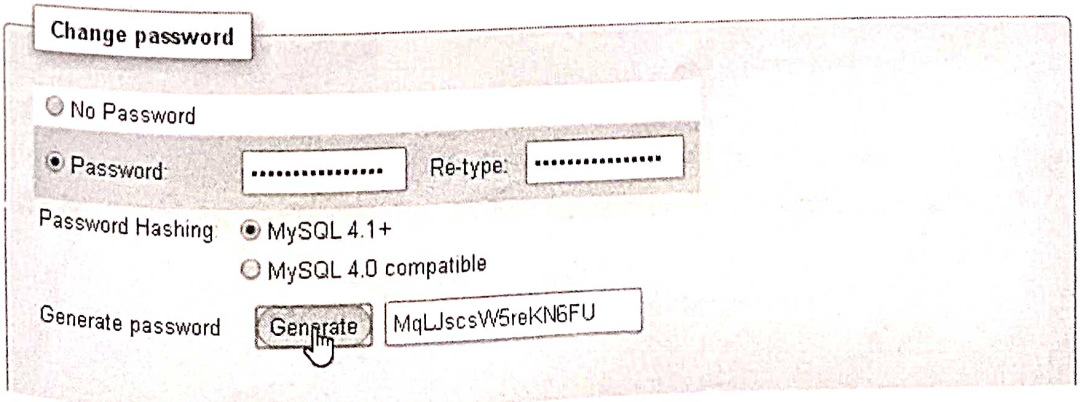
Figure 9-5 MySQL has a comprehensive security system built into it.

You can see that root user is set up (at least in my case) for both localhost and 127.0.0.1, has a password set in both cases, and has all privileges, including the ability to grant privileges to others.

2. Click Edit Privileges on the right end of the first root user listed. The top of the detail privileges page opens. Here you can see the full list of privileges that can be set.



3. Scroll down until you see Change Password, where you can do that if you wish. You can even generate one by clicking Generate to get a 16-character, very strong password created for you. If you do that, be sure to copy the password (select it and press CTRL-C) to a document (open the document and press CTRL-V) that you keep in a secure place. Also, you need to go back to the Zend Server phpMyAdmin entry and use Update to revise your passwords there, or you won't be able to get back into phpMyAdmin.



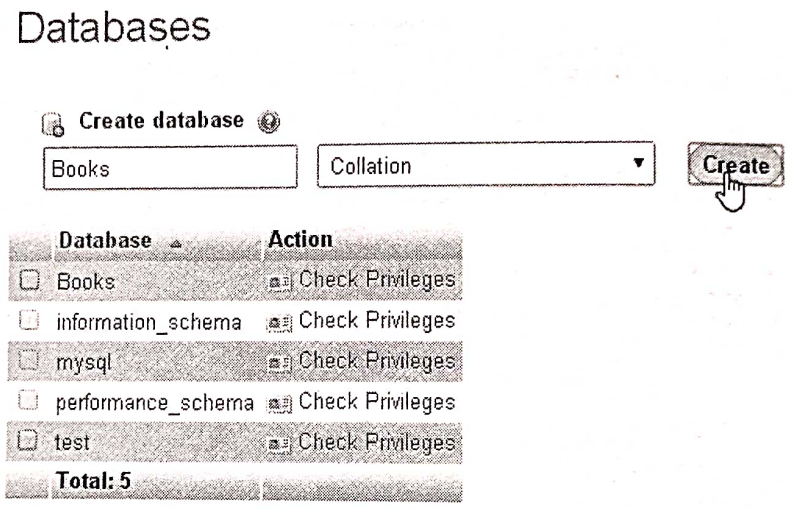
Create and Use a Database in phpMyAdmin

To fully exercise the features of phpMyAdmin, you need to create a database, add a table to it, and add some data to the table. You can then do queries and manipulate the data.

Create a Database and Its Table

Use the following steps to build the full Books database shown earlier in Figure 9-1.

1. In the phpMyAdmin, click the Databases tab and in the Create Database text box, type **Books** and click Create (Collation will be discussed later). A message appears telling you the database has been created.



2. Click Books in the left-hand column to open the new database. Under Create Table in the middle, click in the Name text box, type **Books**, press TAB, and type **6** for the Number Of Columns in the Books table (there is nothing that requires the database and table names to be the same). Click Go. The column entry window opens, as shown in Figure 9-6.

3. In the first field, type **Bkid**, leave INT (integer) for the Type, TAB over to Index, select Primary, and click A_I for Auto Increment. Leave the remaining fields blank.