PHP

UNIT - 1

Chapter 1: PHP Introduction

PHP Basics

Surfing the Web for information can be a boring or a pleasant experience depending on the user friendliness, visual appeal, and interactivity of the Web page. If you have been surfing the Web for quite some time now, you might have noticed that the appearance and functionality of Web pages have undergone a sea change. Gone are the days when a Web page comprised just a few static graphics and some textual information. Today, Web pages have a lot more to offer than just displaying information in a jazzy or eye-catching way. It is the level of interactivity and dynamism that distinguishes one Web page from another and thus affects pages' popularity.

Interactivity and dynamism, in the present-day scenario, are considered the most fundamental features of any Web page besides the obvious visual appeal. To incorporate these features in Web pages, scripting languages today are focusing more on Web-based applications.

> **Note** Scripting languages have been in existence since the 1960s. The first most widely used scripting language was Job Control Language (JCL), which was used in the 1960s to control and arrange data flow in card decks for OS/360. Since then, as a result of technological advancements and ever-changing customer requirements, scripting languages have improved immensely, in terms of both their power and their functionality.

Some of the most common scripting languages that are used today for the Web are Perl, Tcl, Python, and JavaScript. However, one name that stands out among all these scripting languages is PHP. It is a relatively simple language and a much younger one than others of the same class. In this chapter you will learn how PHP scores over other languages and get to know some of the features of PHP that have made PHP the choice of millions of Web developers. PHP has gained widespread popularity and acceptance in the Web-developing community because of its powerful functionality and ease of use.

PHP Primer

PHP stands for Hypertext Pre-Processor. As is suggested by the name, PHP is a pre-processor for hypertext. Being a pre-processor, PHP runs on a remote Web server to process a Web page before it is loaded in a browser. Despite its powerful features, PHP is quite a simple language that has been designed specifically for developing and working with Web pages. Its syntax is similar to that of C and Perl. However, knowledge of both of these languages is not a prerequisite for getting started with PHP, so you can rest easy. It is an open-source software and can be downloaded for free.

One of the foremost reasons for the success of PHP, in addition to other factors such as simplicity and powerful features, is its low entry barrier. Even non-programmers or starters can create their Web sites with ease by using PHP. This probably explains the existence of over five million PHP-based Web sites on the Internet.

I have said much about the popularity of PHP, without discussing some of the fundamental features of PHP that have contributed to this popularity. These features are discussed next.

Fundamental Features of PHP

PHP includes the strengths of relatively older languages, such as Perl and Tcl, and does away with their weaknesses. Although it is known for its advanced features, its fundamental features should be considered first. The advanced features of PHP will be discussed later in the chapter under the topic 'PHP 4: The Latest Version.'

- It is an open source, server-side scripting language.
- It is operating-system independent and thus can be used on any operating system, including Microsoft Windows, Mac OS, Linux, HP-UX, and Solaris, to name a few.
- It supports a wide range of Web servers, such as Apache, Microsoft Internet Information Server, Netscape, and iPlanet.
- It supports a large number of databases, such as MySQL, Ingres, Sybase, Oracle, Base, Informix, FrontBase, and Unix dbm. One of the distinctive features of PHP is that it provides support to database-driven e-commerce Web sites.
- It is simpler to write codes in PHP than in other scripting languages.
- It can be used for creating images, reading/writing files, and sending e-mail. To provide these services, PHP communicates with many protocols, such as HTTP, POP3, SNMP, LDAP, and IMAP.

Now that you're acquainted with the fundamental features of PHP, you will learn about the evolution of PHP.

Evolution of PHP

Rasmus Lerdorf developed PHP in the year 1994. He incorporated this unreleased version of PHP into his home page. He used this version to monitor the number of users who accessed his online resume. No one knew about the existence of PHP until early 1995, when the first version of PHP was released. It was then known as Personal Home Page Tools.

Personal Home Page: PHP's First Released Version

When PHP was first released for developers in early 1995, it was known as Personal Home Page Tools. This version of PHP comprised a *parser* engine that supported a few special macros and some utilities that were commonly used in home pages. This made PHP a popular choice among developers for creating and enhancing their home pages. PHP allowed developers to add some common functionality, such as guest books and counters, to their Web pages.

> **Note** A parser is a program that breaks the source code into an object code. A parser is usually a part of the compiler and it receives input in the form of source program code, markup tags, or online commands. The parser breaks this input to yield objects and methods. These objects and methods can then be managed by other components of the compiler. A compiler is a program that converts programming language into machine language.

PHP/FI: PHP's Second Version

In mid-1995, Rasmus wrote the parser again and renamed it PHP/FI Version 2. The FI in PHP/FI stands for Form Interpreter. This FI was a part of the package developed by Rasmus that could interpret HTML form data. However, PHP/FI did not contain only the Personal Home Page tools scripts and FI. In addition, it included support for mSQL. Thus, the combination of Personal Home Page tools scripts, Form Interpreter, and mSQL resulted in the second version of PHP, or PHP/FI.

Because of its powerful features and built-in database support, PHP/FI did not take long to get noticed, and soon people started working on it to find ways to improve it and add more functionality to it. Such was the popularity of PHP/FI that people even started to contribute their code to it. The rise in popularity of PHP/FI can be gauged from the fact that in late 1996 an estimated 15,000 Web sites used PHP/FI and by mid-1997 this number had swelled to more than 50,000 Web sites.

After PHP/FI came PHP 3, with more powerful and enhanced features.

PHP 3: PHP's Third Version

Until mid-1997, PHP was limited to only a few people who had contributed their code to it-and, of course, to Rasmus, who had conceived it. However, given the speed with which PHP/FI gained recognition and popularity, it soon became the effort of a much bigger, well-coordinated, and systematic team. The parser was completely rewritten from scratch. This new parser, written by Zeev Suraski and Andi Gutmans, formed the very core of the third version of PHP, known as PHP 3. PHP 3 contained a considerable number of new features, and within no time people started using it for developing Web applications.

The combination of PHP 3, Apache, and MySQL soon became an instant hit and was considered the choice for developing Web pages. In addition to improved performance and other features, one more factor that

made this combination an ideal one was that all three technologies were open-source technologies and were freely available. Thus, the problems of obtaining licenses and other hassles were completely eliminated.

Although Apache and MySQL are considered ideal and are the preferred software for use with PHP, this does not mean that PHP is not compatible with other software. It provides an equal degree of support to a variety of Web servers, such as Microsoft Internet Information Server (IIS), Personal Web Server (PWS), iPlanet servers, Oreilly Website Pro server, and Caudium. PHP provides support to a large number of databases. In fact, PHP's support for such a large number of databases is one of its strongest and most significant features. The databases supported by PHP include Adabas, Oracle, Ingres, interBase, FrontBase, Empress, Ovrimos, PostgreSQL, FilePro (read-only), mSQL, Solid, Hyperwave, Direct MS-SQL, Sybase, IBM DB2, Unix dbm, Informix, and Velocis.

The latest version of PHP is PHP 4. This version was introduced in a more complex environment where Web development was not just about writing hard-coded HTML pages. Web applications were needed to be dynamic and interactive, and to provide support for database interactivity. Web applications were developed in such a manner that multiple users could send and retrieve data at the same time. In addition, the time to retrieve data or even load a page was also under consideration. According to a survey based on traditional human factors guidelines, 10 seconds is the maximum response time that the users can wait patiently for a Web page to load; beyond this they tend to lose interest. In addition, if even a single JavaScript error is detected on a Web page, a user is most likely to leave the Web page immediately.

Now you will learn about PHP 4 in detail.

PHP 4: The Latest Version

PHP 4, the latest version of PHP, has a lot of new features that allow it to deliver higher performance and to provide support to an ever-larger range of extensions and libraries. PHP 4 is fast becoming a de facto industry standard for developing Web pages. According to a recent survey conducted by Security Space (`www.securityspace.com`), PHP has emerged as the most popular and widely used scripting language. In October 2001, there were an estimated 1,107,914 PHP users as against 328,856 Perl users, 473,053 Open SSL users, and 1,873 mod_python users. Some of the most common and yet important factors that contributed to the success and rise of PHP are:

- PHP 4 is available for free download from PHP's official Web site. In addition, most software that is used with PHP, such as Apache and MySQL, is also available free of cost.
- PHP is an open-source software. When I say that PHP is open-source software, I do not want to imply that there is any lack of support available for developers who choose PHP for Web development. The advice of several developers who have tried and tested applications using PHP is available free of cost on the Internet.
- A lot of PHP support documents are available free of cost on the Internet.
- Given the enormous number of users who are using PHP, there are numerous mailing lists that you can join. You can then post your problems or take part in discussion forums to solve the problems you are facing.
- There is a dedicated team of volunteer experts who immediately rectify any bugs that are reported or detected in PHP. These experts are the people who are involved with the development of PHP language. They are not affiliated with any organization. If you encounter any bugs in PHP, you can report these bugs at the official site of PHP, **www.php.net/**. This site provides you with all the help you'll need for reporting a bug and contacting the experts.
- In terms of its performance, you can compare PHP with Active Server Pages (ASP). PHP is popular on

Linux platforms, as ASP is popular on Windows. However, cross-platform support provided by PHP makes it score over ASP. PHP can be used as effectively on Windows platforms as on any other platform.

- Unlike many other scripting languages, such as ASP, PHP comes with a built-in compiler that compiles PHP code and can detect errors in it. You can rectify the mistakes in your PHP code based on the mistakes pointed out by the compiler. No such compiler that compiles ASP code is available with ASP.
- Another significant feature of PHP is its portability. PHP is compatible with any combination of software, and this is what makes it portable. It can work with almost any combination of operating system, Web server, and database server.

Now that you're familiar with the success factors of PHP, it's time to review the new and advanced features that have been added in PHP 4.

- Support for Boolean datatype.
- Support for Java and XML.
- Support for COM/DCOM. This support is available only for Windows.
- Support for FTP.
- The '= = =' operator. In addition to checking whether or not the two values are equal, this operator also checks whether the datatypes of these values are the same or not.
- The ability to call a function even before it is declared. This is accomplished by using the runtime binding of functions in PHP 4.
- The PHP highlighter. This feature enables you to view the source code instead of the complete script. This feature helps you to have a faster and better look at the source code.
- Support for variable assignment by reference. This helps you to link two variables so that the value of one variable is dependent on the value of the other variable. Thus, the value of the variable is updated automatically whenever a value is assigned to another variable.
- Server API (SAPI). This feature further enhances the support for Web servers.
- Support for many algorithms, namely Triple DES, MD5, Blowfish, and SHA1. Through the mcrypt library, PHP 4 supports full encryption.
- The ability to reference variables in PHP 4 by using quotes. Additionally, variable expansion is supported using double quotes.

Additional improvements to PHP 4 include:

- In PHP 4, all syntax limitations that existed in PHP 3 have been overcome.
- GET, POST methods in PHP 4 support multi-dimensional arrays.
- In PHP 4, the php.ini file is simple to understand and configure.
- In PHP 4, better ways of creating classes and objects have been introduced.

Introduction to PHP Programming

As discussed earlier, PHP is an open-source, server-side scripting language. What I didn't mention, on purpose, was the fact that it is also an embedded CGI language. This certainly will raise some eyebrows! Now, PHP is an embedded language in the sense that it is enclosed within tags and you can easily switch between PHP and HTML without having to use large amounts of code to output HTML. Common Gateway Interface (CGI) allows you to write computer programs, which can generate HTML and can dynamically process data from Web pages. Before the advent of CGI, you had no option but to create static HTML pages and to keep updating them. Since PHP is a server-side language, you do not need any special browser plug-in or program to run a PHP-enabled Web page. PHP supports all major Web browsers and provides you with tools to build dynamic Web pages.

PHP might be classified as a CGI; however, it imbibes in itself all the characteristics of a complete programming language, such as loop structures, control structures, repetitive tasks, variables, and conditional statements. I will, of course, be dealing with all these features and much more in later chapters.

The three areas where PHP is used are:

- **Server-side scripting:** PHP is most commonly used as a server-side scripting language. To make PHP work as a server-side scripting language, all you need is a PHP parser, a Web server, and a Web browser. However, the Web server has to have a connected PHP installation.
- **Building client-side GUI applications:** Although this is not a preferred use of PHP, with a very good working knowledge of PHP you can build your own client-side GUI applications. You can use PHP-GTK, an extension of PHP, which provides an object-oriented interface for building client-side GUI applications.
- **Command-line scripting:** Your PHP script can run without any browser or server. In this way, it can be used as a simple and easy-to-use scripting language for tasks such as managing databases and messaging files.

After having read so much about the features of PHP, you must by now be quite eager to actually get started with PHP scripting. Without more ado, you may begin. You'll use the customary 'Hello World' example. (Isn't the popularity of this example amazing? This single example has been to software developers what 'Twinkle Twinkle, Little Star' has been to all children.)

The simplest HTML code that you might be aware of goes like this:

```
1.      <html>
2.      <head>
3.      <title> Hello World</title>
4.      </head>
5.      <body>
6.      Hello World
7.      </body>
8.      </html>
```

It can't get any simpler than this!

Now see how PHP can be embedded into this HTML code:

```
1.      <html>
2.      <head>
```

```
3.      <title> Hello World</title>
4.      </head>
5.      <body>
6.      <?php
7.      echo "Hello, World";
8.      ?>
9.      Hello World
10.     </body>
11.     </html>
```

Most of the preceding code is quite similar to the first example discussed, except for the piece of code that is written between line numbers 6 and 8. This piece of code is the actual PHP code.

Tip You can also use just <? and ?> to start and end the code.

Hence, the basic PHP code can be written as:

```
<?php
echo "Hello, World";
?>
```

This produces the output:

```
Hello, World
```

Notice here that instead of writing enormous amounts of code to output HTML, writing only a few PHP commands-just three in the preceding example-does the job.

Chapter 3: Variables, Operators, and Constants

Do you remember how your teachers introduced you to the concept of arithmetic in elementary class? If not, let me refresh your memory a little. Your teacher might have given you a simple example of a basket containing six tomatoes. She might have asked you to put a few more tomatoes in the basket. Then you had to calculate the total number of tomatoes added to the basket, if the final number of tomatoes was eight. For our purposes, you might equate a variable with the basket and an operator with the process of calculating the total number of tomatoes kept in the basket.

In the previous chapters you learned the basics of PHP along with installation and configuration of PHP. In this chapter you will learn about variables, operators, and constants, which are the building blocks of any programming language. The first thing you'll look at will be *variables*. Variables are an important part of writing any code, and you need to understand their use before you begin writing the code. Next you will learn about the different types of *operators* available in PHP and their use. Operators are used by variables or elements to perform calculations and to compare expressions. Finally, you will learn about *constants* and how you can use them effectively while writing your code.

Introduction to Variables

In the previous section I tried to explain the concept of variables by using an example of a vegetable basket. Let me generalize what variables actually are. *Variables* are entities that can hold different values over different periods of time.

Every programming language follows certain rules based on which variables are declared. These rules include the acceptable length of the variables, whether they can contain numeric or alphanumeric characters, whether the name of the variable can include special characters, and whether they can begin with a number. You will now look at the rules that should be followed while naming variables in PHP.

Rules for Naming Variables

In PHP, unlike some other programming languages, there is no restriction on the size of a variable name. In addition, you are free to use both numeric and alphanumeric characters in a variable name. However, there are certain rules that you need to follow. These include:

- Variable names should begin with a dollar ($) symbol.
- Variable names can begin with an underscore.
- Variable names cannot begin with a numeric character.
- Variable names must be relevant and self-explanatory.

Names like `$a` or `$temporary` might be understood and remembered while programming in the short run. However, it would be difficult to find out what they referred to later, when you or other developers look at the code after a long time.

Some examples and non-examples of variable names are:

Examples:

- $prod_desc
- $Intvar
- $_Salesamt

Non-examples:

- $9OctSales
- $Sales123
- $asgs

Now that you have learned about the rules that need to be followed while naming variables, let me explain how to declare variables in PHP.

Declaring and Initializing Variables

Declaring variables implies specifying all the variables that you plan to use within the program. In programming languages like C and C++, these variables have to be declared explicitly along with their datatypes-for example:

```
int Number=10;
```

However, in PHP you do not need to declare the datatype of the variable. You can declare a variable the first time you use it. This is because PHP does not force variables to belong to a specific datatype. The datatype of variables changes based on their content.

Initialization involves assigning values to variables. These values can belong to different datatypes. Datatypes in PHP can be divided into two types, *scalar* and *compound*. Scalar datatypes are simple datatypes that are found in most of the programming languages. These datatypes form the building blocks of a program written in any programming language. Scalar datatypes contain:

- string
- integer
- boolean
- float

On the other hand, combining multiple simple datatypes together and storing them under one name creates complex datatypes. These datatypes are known as compound datatypes, and are used to store data belonging to different datatypes and to manipulate them. Compound datatypes can contain an:

- array
- object

Let me explain these datatypes individually.

- **Integer.** An integer datatype can only contain whole numbers. These numbers can be either positive or negative. You use the ‘-' symbol before the number to represent negative integers.
- **String.** A string datatype is used to store a series of characters. A character is equal to a single byte of

data. You can store 265 different types of characters in a string. However, in PHP there is no restriction on the size of a string.

- **Boolean.** A boolean datatype can only contain either True or False. If you need to refer to the numeric value of a boolean value, True is represented by `1` and False by `0`. This datatype is available only in PHP4.
- **Float.** The float datatype is also used to store numbers. However, float values can also contain decimal numbers. You use the integer datatype to store non-decimal numbers.
- **Array.** An array is used to store multiple values belonging to the same datatype. You can specify the length of an array while declaring it explicitly or at runtime. Each value stored in an array is called an *element*. You reference an element of an array and access its content by referring to its position within the array. You will learn in detail about arrays in Chapter 5, 'Arrays.'
- **Objects.** Objects are the integral part of any programming language. They contain not only data but also its functionality or what needs to be done with the data.

Following are some examples of assigning values to variables:

```
                        $Mystrval = "This is an example of a string value";
                        $Myintval = 145665;
// An integer value
                        $Myboolval = True;
// A boolean value can either be True or False
                        $Myarrval[0] = "My";
//A array containing three elements
                        $Myfloatval=2346.45
// A float value

                        $Myarrval[1] = "First";
//All three elements are referred to by the same
                        $Myarrval[2] = "Array";
// variable name.
```

In the above examples, you are directly assigning values to the variables. You can also assign values to variables by reference. Let me explain in detail about assigning values by reference.

Assigning Values to Variables by Reference

By default, each variable has its own unique value. This means that each variable has an area assigned to it in local memory that it uses to store its current value. Consider, for example, `$empname=Timothy`. Here the system assigns a memory location to the variable `$empname`, where the variable stores the value `Timothy`. If another variable needs to use this value, you need to assign the data explicitly. Therefore, the system needs to maintain multiple copies of the same data, which

leads to data redundancy. You can avoid this by assigning value by reference. In this case, the system maintains only a single copy of the data, which is referenced by all variables that require the value. Therefore, all the variables access the same area in the memory.

You can assign a value by reference by putting an ampersand (`&`) symbol in front of the variable name. This assigns the memory location of the data to the variable instead of creating another copy of the data.

Now that you know the rules that should be followed while specifying a variable name, consider the different scopes of variables. The *scope* of a variable in a program is the area where you can reference the variable

directly.

Scope of Variables

The scope of a variable in a program determines whether other programs can access the variable from outside the parent program or whether parts of the same program can access the variable. The parent program is the program in which the variable is defined and initialized. The scope of a variable also determines when the system creates the variable in the memory and when it destroys it.

If you declare a variable inside a function or as an argument to a function, the scope of the variable is limited to *local* scope. This means that the variable can only be assigned a value or used inside the function in which it is declared. However, if you declare the variable outside the function, the scope of the variable is *global*. This means that you can use the variable or manipulate its value in any function.

> **Note** You use a *function* to store a piece of code together and under one name. This provides easy reusability since you only need to specify the function name whenever you need to reuse code. Using functions also helps in debugging the code, as you only need to make a change in one place. Any change made to the code is cascaded across all references of the function. This makes the code easy to follow and understand. You will learn more about functions in Chapter 6, 'Functions.'

In PHP, you can have either local or global variables. The scope of any variable declared within a function is by default limited to local. In PHP, a variable can be global by declaration, but it is still limited to local scope if you do not assign a value to it outside the function. In PHP, you need to declare variables as global and initialize them inside the function where they are used. This is unlike C, where a variable once defined as global is accessible within all functions. A function can contain an unlimited number of global variables. You will learn more about the variables in Chapter 6, in the section 'Variable Functions and Variable Argument Functions.'

Environment Variables

You might already know that most scripts or certain types of program codes execute on the client browser. However, certain types of scripts execute on a Web server; these scripts are known as *server-side scripts*. These server-side scripts use *environment variables* to pass information stored on a Web server to external programs requesting the information. These variables store information regarding the server, such as access and logon information. Although these variables are called environment variables, they are not related to the variables managed by the operating system. These variables manipulate information on the Web server. The only time these variables are used by the operating system is while passing information back to the program that had requested the information.

You can also set values for environment variables. Some of the common environment variables available in PHP and their functions are shown in Table 3-1.

Table 3-1: PHP Environment Variables and Their Functions

Variable name	Function
`$argv`	Contains all the arguments passed to a script from the command table text.
`$argc`	Contains a count of the number of arguments passed to a script from the command line.

Variable name	Function
$PHP_SELF	Contains the name of the currently executing script. However, it is not available if PHP is run from the command line.
$HTTP_GET_VARS	Contains an array of variables retrieved by using the HTTP GET method and is stored in the current script.
$HTTP_POST_VARS	Contains an array of variables retrieved by using the HTTP POST method and is stored in the current script.
$HTTP_COOKIE_VARS	Contains an array of variables retrieved by using HTTP cookies and is stored in the current script.
$HTTP_ENV_VARS	Contains an array of variables stored in the current script by using the parent environment.
$HTTP_POST_FILES	Contains an array of variables that have information regarding uploaded files. By using the HTTP POST method, you can upload these files.
$HTTP_SERVER_VARS	Contains an array of variables passed to the current script by the HTTP server.

Now that you know about variables, let me tell you how you can further manipulate these variables. Manipulating variables involves performing mathematical, assignment, comparison, and concatenation operations on them. In order to manipulate variables, you need to use operators to perform these operations. PHP provides different types of operators to perform different tasks. The following section discusses these operators in detail.

Operators

An *operator* is a part of code that is used by one or more variables or elements to perform calculation and to compar expressions. You can use an operator to perform arithmetic, concatenation, comparison, and logical operations.

PHP provides the following operators as shown in Table 3-2 :

Table 3-2: Operators Available in PHP

Operator	Used for
Arithmetic operators	Are used for mathematical calculations.
Assignment operators	Are used for assignment operations.
Comparison operators	Are used for comparisons.
Execution operators	Are used for executing the stored code as a shell command and assigning the output to a variable.
Increment/decrement operators	Are used for incrementing and decrementing values.
String operators	Are used for concatenating two expressions.
Concatenation operators	Are used for combining strings.

Now you'll learn about these operators in detail.

Arithmetic Operators

As you might have guessed from the name, you use arithmetic operators to perform mathematical calculations. Table 3 lists the arithmetic operators available in PHP.

Table 3-3: Arithmetic Operators Available in PHP

Operator	Name	Purpose
+	(Addition)	Calculates sum of two integers or concatenates two strings.
-	(Subtraction)	Calculates the difference between two integers.
*	(Multiplication)	Calculates the product of two integers.
/	(Division)	Calculates the quotient of two integers.
%	(Modulus)	Calculates the remainder obtained by dividing two integers.

Consider the following examples of arithmetic operators:

```
$Num1 = 9;
$Num2 = 3;
$Result = $Num1 + $Num2;        // Output $Result == 12
$Result = $Num1 - $Num2;        // Output $Result == 6
```

```
$Result = $Num1 * $Num2;        // Output $Result == 27
$Result = $Num1 / $Num2;        // Output $Result == 3
$Result = $Num1 % $Num2;        // Output $Result == 0
```

In the above example, variables $Num1 and $Num2 are assigned values and arithmetic operators are used to add, subtract, multiply, and divide these variables. For example, in the first case the value stored in the variable $Num1 is added to the value stored in the variable $Num2 and the output is stored in a variable $Result .

The Multiplication (*) Operator

The multiplication operator is used to calculate the product of two numbers. The syntax for the multiplication operator given below:

```
$Result=$Num1*$Num2;
```

In the above syntax, PHP considers the datatype of $Num1 and $Num2 as numeric. The multiplication operator calculates the product of $Num1 and $Num2 .

In the following examples, the multiplication operator is used to calculate the product of two integer values:

```
$Result=3*5; // Returns a value of 15
$Result=556.26*262.34; //Returns a value of 145929.2484
```

The Division (/) Operator

You use the division operator to divide two numbers. The operator returns the result as a float. The syntax for the division operator is given below:

```
$Result=$Num1/$Num2;
```

In the above syntax, $Num1 and $Num2 are numeric variables. The / operator returns the quotient of $Num1 divided $Num2 as a float number.

The following examples show the use of the / operator to divide two numbers.

```
$Result=19/3;                                   //Returns a value of
6.3
$Result=26/3;                                   //Returns a value
of 8.66667
```

In the above examples, the first integer value is divided with the second integer value and the result is stored in the variable $Result .

The Modulus (%) Operator

You use the modulus operator to divide two numbers and retrieve the remainder. Following is the syntax of the modu operator:

```
$Result=$Num1%$Num2;
```

In the syntax, $Num1 and $Num2 are two integer values. The modulus operator returns the remainder left after dividir $Num1 and $Num2 .

Following are a few examples in which the % operator is used:

```
$Result=24%8;                                    //Returns 0
$Result=18%8;                                    //Returns 2
$Result=12%4.3;                                 //Returns 3.4
```

```
$Result=47.9%9.35;                                    //Returns 1.15
```

Note If you use float variables as operands along with the modulus operator, then the remainder will also be a float variable.

The Addition (+) Operator

You use the addition operator to concatenate two strings or to add two numbers. Following is the syntax for the addit operator:

```
$Result=$Variable1+$Variable2;
```

The variables used in the expression can be either integers or strings. If you use integer values, the addition operato returns the sum of `$Variable1` and `$Variable2` . However, if the variables are strings then the result is a concatenated string.

Caution While using the addition operator, you must ensure that both variables are of the same datatype.

Consider the following examples:

```
$IntValue=8+6;
//Returns 14
$IntValue=348.45+5468;
//Returns 5816.45
$StrValue="Hello"+ "World";
//Returns Hello World
```

If the operands are integers or floats, as is the case of the above two examples, operators add the numbers to each other and return an integer or float value respectively. In cases where the operands are strings, the operator concatenates both the strings to each other.

The Subtraction (-) Operator

You can calculate the difference between two numbers by using the subtraction operator. The syntax of the subtracti operator is as follows:

```
$Result=$Num1-$Num2;
```

In the above syntax, `$Num1` and `$Num2` are two integer values. Consider the following examples:

```
$IntResult=15-4;
//Returns 11
$IntResult=756.45-325.26;
//Returns 431.19
```

Assignment Operators

The moment you see the = operator you probably associate it with "equal to." Though this might be true in the case o other languages, in the case of PHP it is an *assignment* operator and you use it to assign a value to a variable.

For example:

```
$Myvar=12;
//$Myvar
becomes an integer type
```

```
variable and contains the value 12.
                $Myvar="World";
//$Myvar
becomes a string type variable

and now contains the value ‘World'.
```

PHP also allows the use of more that one assignment operator in a single statement. For example:

```
                $Result=($Num1=3)+($Num2=1);
//$Result,$Num1, and $Num2 are integer datatype

variables and hold the value 4,3,and 1, respectively.
```

In the above example, first `$Num1` and `$Num2` are assigned values, then their sum is calculated, and finally the result stored in `$Result` .

PHP also supports the combined arithmetic operators += and `.=` . These operators not only initialize a value to an expression but also store the result obtained from executing the expression. The += operator is used for numeric typ data while the `.=` operator is used for string type data.

For example:

```
                $Num=4;
                $Num+=1;
//Increments the value
of $Num by 1
                echo $Num;
//Result 5
```

Note that you can also write the expression `$Num+=4` as `$Num= $Num + 4` . In the above example, the variable `$N` is assigned a value and in the next step `1` is added to the current value of `$Num` and the result is assigned back to th variable.

```
                 $Myvar="Add";
                $Myvar .=" Something";
//$Myvar=="Add Something"
```

The above expression can also be written as `$Myvar = "Add". "Something"` or even as `$Myvar = "Add Something"` . The `.=` operator appends the new value with the current value of `$Myvar` .

Comparison Operators

You use the *comparison* operators to compare expressions. You can compare whether the values of two expressions are either more than, less than, or equal to one another. Table 3-4 lists the comparison operators used in PHP.

Table 3-4: Comparison Operators Available in PHP

Name	Symbol	Purpose
Equal	==	Returns True if the value of one variable is equal to another.
Identical	===	Returns True if the value of one variable is equal to another and both are of the same datatype.

Name	Symbol	Purpose
Not equal	!=	Returns True if the value of one variable is not equal to the other.
Less than	<	Returns True if the value of one variable is less.
Greater than	>	Returns True if the value of one variable is more.
Less than or equal to	<=	Returns True if the value of one variable is less than or equal to the other.
Greater than or equal to	>=	Returns True if the value of one variable is more than or equal to the other.

The ternary operator ? is another conditional operator that is available in PHP. You might have used this operator in and many other languages to compare two expressions and display a third expression if both the expressions match. The syntax of the ternary operator is shown below:

```
(expr1)?(expr2):(expr3);
```

In the above example, two variables $Num1 and $Num2 are compared and checked if they are equal to each other. T datatypes of the variables are also checked by using the comparison operators (== , === , != , < , > , <= , >=). The expressions return the boolean value True or False based on their result.

```
<?php
        $Num1 = 13;
        $Num2 = 12;
        echo $Num1," == ", $Num2, " = ", $Num1 == $Num2, "<br>\n";//
Returns False

        $Num1 = 12.0;
        $Num2 = 12;
        echo $Num1," == ",  $Num2, " = ", $Num1 ==  $Num2, "<br>\n";
        // Returns True because == does not compare the datatypes,
only compares the values

        $Num1 = 12;
        $Num2 = 12;
        echo $Num1," == ",  $Num2, " = ", $Num1 ==  $Num2,
"<br>\n";// Returns True
?>
```

The output of the above code is shown in Figure 3-1 .

```
X-Powered-By: PHP/4.0.4pl1
Content-type: text/html

13 == 12 = <br>
12 == 12 = 1<br>
12 == 12 = 1<br>
```

Figure 3-1: Output of comparison operators

In the above examples, the values of $Num1 and $Num2 are compared to each other by using the == operator. In the

first case, since the value of $Num1 is not equal to the value of $Num2 , the expression returns False. In the second case, the expression returns True because the values of both the expressions match even though their datatypes dif Similarly, the third expression also returns True.

Consider another example.

```
<?php
                $Num1 = 13;
                $Num2 = 12;
                echo $Num1," == ", $Num2, " = ", $Num1 == $Num2, "<br>\n";//
Returns False

                $Num1 = 12.0;
                $Num2 = 12;
                echo $Num1," == ",  $Num2, " = ", $Num1 ==  $Num2, "<br>\n";
                // Returns True because == does not compare the datatypes,
only compares the values

                $Num1 = 12;
                $Num2 = 12;
                echo $Num1," == ",  $Num2, " = ", $Num1 ==  $Num2, "<br>\n";
// Returns True
        ?>
```

The output of the above code is shown in Figure 3-2 .

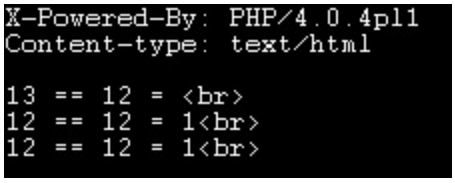

Figure 3-2: Another program using comparison operators

In the above examples, the values of $Num1 and $Num2 are compared to each other by using the === operator. The === operator not only compares the values of variables but also their datatypes. In the first case, even though the va of $Num1 and $Num2 are the same, their datatypes are different. Therefore, the expression returns False. Similarly, t second expression also returns False since one variable is a string datatype and the other an integer. The third expression returns True because the values and the datatypes of both the variables are the same.

```
        <?php
                $Num1 = 12;
                $Num2 = 12.0;
                echo $Num1," != ",  $Num2, " = ", $Num1 !=  $Num2, "\n";//
Returns 12 != 12 False

                $Num1 = 12;
                $Num2 = 13;
                echo $Num1," != ",  $Num2, " = ", $Num1 !=  $Num2, "\n";//
Returns 12 !=13 True
                echo $Num1," < ",   $Num2, " = ", $Num1 <   $Num2, "\n";//
Returns 12<13 True
                echo $Num1," > ",   $Num2, " = ", $Num1 >   $Num2, "\n";//
Returns 12 > 13 False
```

```
                echo $Num1," <= ",  $Num2, " = ", $Num1 <=  $Num2, "\n";//
Returns 12<=13 True
                echo $Num1," >= ",  $Num2, " = ", $Num1 >=  $Num2, "\n";//
Returns 12 >=13 False
         ?>
```

In the above examples, the values of the variables `$Num1` and `$Num2` are evaluated by using the comparison operat and their result is displayed.

In the next section , you'll learn about the third type of operator-the execution operator.

Execution Operator (` `)

In PHP, only one type of execution operator is available, the backticks (` `) operator. You use the execution operat to implement the code stored in the backticks as a shell script and assign the output to a variable. Shell scripts are similar to programs and are used to execute multiple commands together. The variable, in which the output is stored used to manipulate the data since the output is not dumped on the HTML page. You can manipulate the output and display it in the format you want.

For example,

```
         $allfiles=`ls`;
         echo"<pre>$allfiles</pre>"
//Pre is the html tag for
pre formatted output.
```

In the above output, the LINUX command `ls` is executed and its result is stored in the variable `$allfiles` . You ca now manipulate the content of the variable `$allfiles` and display specific files based on certain conditions.

Increment and Decrement Operators

In the previous section , you learned how to increment the value of variables by using the arithmetic operators. Thou this is the conventional method, it is better suited to cases of two different operands. In the case of a single operand, the expression is unnecessarily longer since the same variable is repeated twice-for example, `$Num=$Num+1` or `$Num=$Num-1` . Instead of the arithmetic operators, you can use the `++` and `--` operators, also known as the *unary operators* . In the case of unary operators, the variable is incremented and the incremented value is then reassigned back to the variable.

You can use the unary operators in two ways. You can either increment the value of a variable or decrement it. The assigned incremented/decremented value of the variable depends on whether you write the operator at the beginnin the end of the variable. When the `--` operator is at the beginning of the variable, the variable is first decremented an then the decremented value is reassigned to the variable. Otherwise, if the `--` operator is at the end of the variable, current value is returned and then the variable is incremented. Table 3-5 lists the various increment and decrement operators.

Table 3-5: Increment and Decrement Operators

Name	Operator	Purpose
Pre-increment	`++$myvar`	Increments the current value of `$myvar` by `1` , then returns the new value of `$myvar` .
Post-increment	`$myvar++`	Returns the current value of `$myvar` , and then increments the current value of `$myvar` by `1` .

Name	Operator	Purpose
Pre-decrement	`--$myvar`	Decrements the current value of `$myvar` by `1` , then returns the new value of `$myvar` .
Post-decrement	`$myvar--`	Returns the current value of `$myvar` , then decrements the value of `$myvar` by `1` .

For example:

```
        <?php
                    $Num = 17;
                    echo "Num++: " . $Num++ . "<br>\n";
// Current value 17 (Post Increment)
                    echo " Num1: " . $Num . "<br>\n";
// Current value 18
                    $Num = 17;
                    echo "++Num: " . ++$Num . "<br>\n";
// Current value 18 (Pre Increment)
                    echo " Num: " . $Num . "<br>\n";
// Current value 18
                    $Num = 17;
                    echo "Num--: " . $Num-- . "<br>\n";
// Current value 17 (Post Decrement)
                    echo " Num: " . $Num . "<br>\n";

// Current value 16
                    $Num = 17;
                    echo "--Num: " . --$Num . "<br>\n";
// Current value 16 (Pre Decrement)
                    echo "  Num: " . $Num . "<br>\n";
// Current value 16
        ?>
```

The output of the above code is shown in Figure 3-3 .

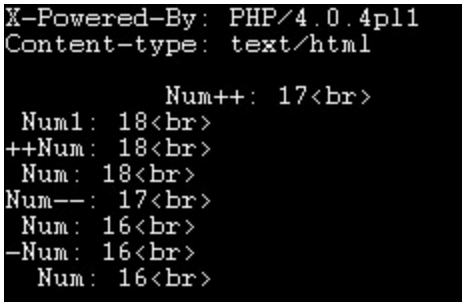

Figure 3-3: Output of the increment/decrement operators

In the preceding example, the value of variable `$Num` is first incremented and then decremented by using the pre- an post-incremental and decremental operators. As you might have noticed, when you use the post-incremental operato the current value of `$Num` , which was `17` , is first displayed and then incremented. The current value of `$Num` is now . However, when you use the pre-incremental operator, the current value of variable `$Num` is first incremented and th the value is assigned back to the variable. Therefore, the value `18` is displayed twice. Similarly, when you use the pre

decremental operator, the current value of the variable, which was `17` , is first displayed and then the value is decremented. The current value of the variable is now `16` . Finally, in the case of post-decremental operator, the valu of `$Num` is first decremented and then the old value is assigned to the variable. Therefore, in both cases the value 16 displayed.

String Operators (.)

There are two types of *string operators* available in PHP. The `.` , or *concatenation operator* , concatenates two expressions and forms a new string. The other is the `.=` , or *concatenation assignment operator* , which concatenate the expression specified on the right side with the one on the left.

See, for example:

```
          <?php
                    $Myvar1 = "some";
                    $Myvar2 ="thing";
                    $concat = $Myvar1 . $Myvar2;
                    echo $concat ,"\n";
// $concat1=="something"

                    $Myvar1 = "some";
                    $Myvar1 .= "thing";
                     echo $Myvar1,"\n";
// $Myvar1=="something"

                    $Myvar1 = "24 cats" +5;
                    echo $Myvar1 ,"\n";
//Returns 29

                    $Myvar1 = "24cats" + 5;
                    echo $Myvar1 ,"\n";
//Returns 29


                    $Myvar1 = "cats 24" + 5;
                    echo $Myvar1 ,"\n";
//Returns 5

                    $Myvar1 = "24 cats" . 5;
                    echo $Myvar1 ,"\n";
//Returns 24 cats5

                    $Myvar1 = "24cats" . 5;
                    echo $Myvar1 ,"\n";
//Returns 24cats5

          ?>
```

The output of the above code is shown in Figure 3-4 .

Figure 3-4: String manipulation by using the string operator

In the above example, the string operators are used to concatenate strings and integer values. As you might have noticed, the two strings (`some` and `thing`) are combined together to form a single string (`something`). However, i cases of concatenating a string and an integer, if the string begins with numeric characters, the result is the sum of tl integer values. If the numeric characters are in the middle of the string, only the numeric variable is displayed. If you the `.` operator, the integer variable is also treated as a string and both the string and the integer are concatenated.

Now that you have learned about operators and how they are used to work with variables, you'll next learn how you c manipulate string. In the next section , you'll also look at changing the datatype of variables.

String Manipulation

You need to know how to manipulate strings in order to work effectively with any scripting language. You manipulate strings to extract information from HTML pages. For example, you can dynamically create unique user IDs by extract the user's name from the details form and create a user ID with it. PHP provides certain functions to help in string manipulation. Let me now describe these functions and their use in detail.

The substr Function

The `substr` function extracts a part of a string. It uses three parameters-the string, the start position, and the numbe of characters to retrieve. The start position starts at zero and the length parameter is optional. The syntax of the `substr` function is given below.

```
string substr(string string, int start [, int length])
```

If the value of the `start` position is positive, the extraction starts from the first character in the string. However, if the `start` position is negative, the starting point of extraction is calculated from the end of the string.

For example:

```
$extract = substr("Hello",1);
// Returns "ello"
$extract = substr("Hello",1,2);
// Returns "el"
$extract = substr("Hello",-2);
// Returns "lo"
$extract = substr("Hello",-3,2);
// Returns "ll"
```

If the length parameter is available and is a positive value, the specified number of characters is retrieved. However, the length is negative, then the extracted string contains the specified number of characters until the end of the string for example:

```
$extract = substr("Hello",1,-1);
```

```
// Returns "ell"
                    $extract = substr("Hello",2,-2);
// Returns "l"
```

The substr Function

The `substr` function finds the first occurrence of a search string in a given string. The function returns the value Fal
if the string is not found.

The function is case sensitive and its syntax is as follows:

```
                    string strstr(string search_string, string
base_string)
```

For example:

```
                    $content="Motherboard";
                    $base=strstr($content, ‘b');
                    echo $base;
// Returns ‘board'
```

In the above example, the string `$content` is searched for the character ‘b ' and when the character is found, the re
of the string from the character to the end of the string is extracted.

The str_replace Function

The `str_replace` function searches in a base string for all instances of a search string and then replaces it with
another string. In cases where the size of the replace string is smaller than the search string, spaces are appended a
the end of the replacement string. The function's syntax is as follows.

```
                    mixed str_replace(mixed search_string, mixed
replace_string, mixed base_string)
```

For example:

```
                    $story = "This is an excellent example of a horror story.";
                     $retstring = str_replace("excellent", "exceptional", $story);
```

In the above example, the string '`excellent` " is searched for in the text stored in the variable `$story` and if it is
found, it is replaced by the string "`exceptional` ". All instances of the search string are found and replaced. The fin
content of the variable `$story` is "`This is an exceptional example of a horror story.` "

In the next section , you'll learn about type juggling and how it helps you to program in PHP.

Type Juggling

You do not need to declare variables in PHP to belong a specific datatype; their datatype changes based on their content. For example, a variable called `$Test_item` can initially contain a user's name, which is of string type, then it can store the user's age, which is an integer type, and finally it can also store the user's salary, which is of float type. Therefore, in PHP, a variable can store any kind of data without it being explicitly declared of the specific datatype. You can also store a numeric value as string, if you assign it within a pair of quotes.

Following are some examples of type juggling.

```
$Myval = "1500";              //Stores the value as a string
$Myval=1500;          //Stores the value as an integer
$Myval="14567.57";    //Stores the value as a string
$Myval=14567.57;      //Stores the value as a float
```

In the above example, the datatype of `$Myval` changes based on the value you assign to the variable.

Besides changing the datatypes of variables based on their content, you can also explicitly assign a datatype to a variable by using type casting.

Type Casting

PHP also handles the functionality of type casting in the same way that languages such as C handled it. In type casting, you explicitly change the datatype of a variable from one type to another. In PHP, you can implement the following types of type casting, as shown in Table 3-6.

Table 3-6: Categories of Type Casting

Purpose	Datatype
To convert a value to boolean	(bool), (boolean)
To convert a value to integer	(int), (integer)
To convert a value to float	(float), (double), (real)
To convert a value to array	(array)
To convert a value to object	(object)

You can also convert a variable to a string by enclosing it within double quotes. When you cast an array to a string, the result is the string 'Array'. However, if you cast an object to a string, the result is the string 'Object'. On the other hand, if a string is converted to an array, the string forms the first element of the array. In a case where you convert a string to an object type, the result is an attribute called ‘scalar', with the variable as its attribute.

For example:

```
                    $Val = ‘Hello';
                    $Greet = (array) $Val;
                    echo $Greet [0];
//Result ‘Hello'

                    $Val = ‘Hello';
                    $obj = (object) $Val;
                    echo $obj->scalar;
//Result ‘Hello'
```

In the above examples, the variable `$Val` contains the value `Hello`. In the first example, the variable `$Val` is type casted into an array variable named `$Greet`. This variable stores each character of the word as separate elements of the array. In the second example, the same variable `$Val` is type casted into an object variable named `$obj`. You can also use this object to display the value.

You can convert the datatype of any variable to either boolean, integer, or string. However, you need to follow certain rules while converting a variable from one datatype to another. The following sections discuss the conversion of variables to boolean, integer, and string datatypes.

Boolean Conversion

PHP 4 provides the functionality to convert a value of a different datatype to boolean. A boolean value is derived from an expression and can be either True or False.

You can use either of the (bool) or (boolean) typecasts to convert a variable datatype to boolean datatype. However, implicit conversion of the value to boolean datatype is more common in PHP if the operator or the function needs a boolean argument.

While converting a value to boolean datatype, in the following conditions the boolean value will be False:

- String variable contains `0` or is empty.
- Integer variable is `0.`
- Variable contains the value NULL.
- Array contains `0` elements.
- Object contains `0` elements.

In all other cases, the boolean value is True.

> **Note** Here it is important to note that even -1 is converted to False since it is a non-zero value.

Integer Conversion

You use the typecasts (int) or (integer) to convert a value to an integer datatype. However, automatic type conversion is more common. If you try to convert a float value to an integer, the value is rounded off to the closest value near zero. If you convert a boolean value to an integer, a True value is converted to `1` and a False is converted to `0`. As you will see in cases of string conversion, in string variables the initial part of the string is taken into consideration while determining how it will be converted.

String Conversion

A string that contains a numeric value can easily be converted to an integer. However, if the string contains the characters `.`, `e`, or `E`, it is converted into a float value. The initial section of the string is taken into consideration when determining the datatype of the converted string. For example, if the string begins with a numeric value, the converted value is numeric, and if the string ends with a numeric value, the converted value is `0`. An exception to this rule is in cases where the string contains both alphabets as well as numeric values but with a space between them. In this case, the converted value will only be the numeric part.

For example,

```
$SalesItem = "15 boxes grain" + 2;
//Result is
17
$SalesItem = "15units" + 2;
//Result is 15
$SalesItem = "12.4" + 2;
//Result is 14.4
$SalesItem = "-12.4kg" + 2;
//Result
is -14.4
```

In the above examples, the addition operator is used to add two types of values; one is a string and the other an integer. As you might have noticed, if the string value contains numeric characters, the characters are extracted and added to the numeric value.

In the following section, you'll learn about a unique concept in PHP, known as variable variables.

Variable Variables

A *variable variable* creates a new variable and assigns the current value of a variable as its name. Though this may sound a little confusing, what it means is that the current value stored in variable `$greet` is used to create another variable. This is explained in the following example.

```
                    $greet= "greeting";
                    $$newgreet= "hello!";
//This is the same as writing $greeting="hello!";
```

In the above example the variable `$greet` contains the value `greeting`. Instead of creating another variable called `$greeting` and assigning a value to it, you can use `$$newgreet`. This creates a new variable based on the current value of `$greet` and assigns the value `hello` to it. In this way, you can create dynamic variables and assign values to them.

Similarly, any data stored in a variable can also become a function name. In the example shown below, the value `greeting` is assigned to the variable `$func`. Since `greeting` is an already defined function, writing `$func( )` in your code is the same as referencing the function `greeting()`. You'll learn more about variable variables in Chapter 6, "Functions."

As discussed before, variables in PHP are declared only when they are used and do not need to be defined as being of a specific datatype. However, sometimes you might need to assign a datatype to a variable. This is common in cases where you want to force the variable to store data of only a specific datatype. In such cases, you need to explicitly set the variable to belong to a specific datatype. PHP provides two functions, `settype()` and `gettype()`, to do this. The `gettype()` function retrieves the current datatype of a variable and the `settype()` function assigns the variable to a specific datatype. Now I will explain these functions in detail.

Functions for Determining and Setting Variable Types

Although in PHP you usually do not need to declare the datatypes for variables, in certain cases it becomes necessa so. For example, you might need to dynamically set the datatype of a variable based on some output. To do so, you to find out its current datatype and set a new datatype. PHP provides two functions for determining and setting the da variables.

The `settype()` Function

This function forcibly converts the datatype of a variable to another. The syntax of `settype()` is as follows:

```
boolean settype (mixed $myvariable,
string set
datatype)
```

The `settype()` function converts the datatype of `$myvariable` to the value of `setdatatype` . The return type is and returns the value True if the conversion is successful. If it is not successful, the value False is returned.

The value `setdatatype` can belong to any of the following datatypes: integer, string, boolean, float, null, array, and

For example:

```
$Mystrval = "Container 72";
//String value
$Myboolval=False;
//Boolean value
settype($Myboolval, "string");
//Returns 0
settype($Mystrval, "integer");
//Returns 72
settype($Myboolval, "integer");
//Returns 0
```

In the above example, the variables `$Myboolval` and `$Mystrval` contain boolean and string values, respectively. the `settype()` function to explicitly convert the datatype of `$Myboolval` to string or integer datatypes and the valu `$Mystrval` to an integer datatype.

The `gettype()` Function

The `gettype()` function retrieves the datatype of a variable. The syntax is as follows:

```
String gettype (mixed $mixedvar)
```

The datatype of the variable `$mixedvar` is retrieved and returned as a string. The string can be any of the following datatypes: integer, string, boolean, double, null, array, object, or resource.

For example:

```
<?php
$Myval = "10 pigs" +2;
echo $Myval;
//Returns 12
```

```
                        echo gettype($Myval),"\n";
//Returns integer
                        settype($Myval,"double");
                        echo $Myval ;
//Returns 12
                        echo gettype($Myval),"\n";
//Returns double
                        $Myval = "10pigs" + 2;
                        echo $Myval ;
//Returns 12
                        echo gettype($Myval),"\n";
//Returns integer
                        $Myval = "10 pigs" . 2;
                        echo $Myval ;
//Returns 10 pigs2
                        echo gettype($Myval),"\n";
//Returns string

                        settype($Myval,"integer");
                        echo $Myval ;
//Returns 10
                        echo gettype($Myval),"\n";
//Returns integer

                        $a = "10pigs" . 2;
                        echo $Myval ;
//Returns 10pigs2
                        echo gettype($Myval),"\n";
//Returns string

                        settype($Myval,"integer");
                        echo $Myval ;
//Returns 10
                        echo gettype($Myval),"\n";
//Returns integer

                        settype($Myval,"boolean");
                         echo $Myval ;
//Returns (1/true)
                        echo gettype($Myval),"\n";
//Returns boolean

                        $Myval ="";
                        settype($Myval,"boolean");
                        echo $Myval ;
//Returns (blank/NULL/FALSE)
                        echo gettype($Myval),"\n";
//Returns boolean
                        ?>
```

In the above examples, the datatypes of the variables are retrieved and displayed. You can also explicitly set the dat the variables by using the `setdatatype()` function and display the datatype by using the `getdatatype()` functio

Constants

Suppose you need to use a particular value throughout an application. For example, say you need an application that calculates and displays the highest marks attained by individual students on a test. To calculate this, the application needs to store the highest marks at a single location, since it needs to be accessed repeatedly. In such a case, instead of creating multiple instances of the same data every time, you can use *constants*. A constant is a space in the memory where you can store values that do not change during the execution of a program.

Defining Constants

A constant has the following properties:

- It is case sensitive and can be in either uppercase or lowercase.
- It follows the same naming convention as a variable.
- It can only begin with an alphabet or an underscore followed by any number of alphanumeric characters.
- It, by default, has global scope.

To define a constant you use the `define()` function. Once a constant is defined, it cannot be changed. A constant can have four types of values: string, boolean, double, or integer.

Unlike a variable, you can directly access a constant by its name instead of appending a `$` symbol before it. You can use the `constant()` function in case you need to dynamically retrieve a constant's value in a program. You use the `get_defined_constants()` function to retrieve a list of all the currently defined constants.

Consider the following statement:

```
<?php
define ("CONSTANT", "This cannot be changed");
echo $CONSTANT;
// Returns This cannot be changed
echo $Constant;
// Returns Constant
echo "CONSTANT";
//Returns CONSTANT
echo 'CONSTANT';
// Returns CONSTANT
?>
```

In the above example, you created a constant variable and named it `CONSTANT`. You also stored the value "`This cannot be changed`" in it. Since constants are case sensitive by default, the name `CONSTANT` is different from `Constant`. Therefore, after the execution of the second echo statement, the compiler first searches for a value in the *predefined constants*. Since it does not find the value, it then searches for the value in the *user-defined constants*. As it still does not find the value, it just prints the value. However, PHP will produce a notice for such an operation. PHP assumes all values outside double quotes and without a dollar sign are constants and prints the value. In other words, you cannot print constants within quotes.

Although constants are by default case sensitive, you can use the define function to make them case insensitive. Observe the example given below.

```
define("CONSTANT", "This cannot be changed.");
//Creates a case-sensitive constant.
define("CONSTANT", "This cannot be changed.", 1);
//Creates a
case-insensitive constant.
```

In the above example, when you add the third parameter `1` to the `define()` function, the variable `CONSTANT` becomes case insensitive. Now you'll consider some of the predefined constants available in PHP.

Predefined Constants

Just as you have predefined variables, you also have predefined constants in PHP. You can use these constants to retrieve information regarding the PHP. These constants are listed in Table 3-7.

Table 3-7: Predefined Constants Available in PHP

Constant name	Function
PHP_VERSION	Retrieves the version of PHP currently in use.
__FILE__	Retrieves the name of the file currently being parsed. If the constant is used within an included file, then the file name is retrieved and not the name of its parent file.
__LINE__	Retrieves the total number of lines in the current parsed file. If the constant is used within an included file, then the number of lines in the included file is given.
E_ERROR	Indicates errors except for parsing errors from which it is not possible to recover.
E_PARSE	Indicates that the parser was stuck in a syntax error in the script and recovery is not possible.
E_WARNING	Indicates that a warning is generated about an error encountered by PHP but the program can still work.
E_ALL	Stands for all the E_* constants combined.
E_NOTICE	Indicates something that you should notice that is not an error.
TRUE	Indicates a TRUE value.
FALSE	Indicate a FALSE value.
NULL	Indicates a NULL value.

Alternative Syntax for HTML-Embedded PHP Code

This syntax is not very popular, but you still need to use it when you require static HTML content along with the dynamic PHP content. This syntax is common for all control structures. The opening braces of the control structure change to a colon (:) and the closing braces are replaced by `endfor`, `endwhile`, `endif`, `endforeach`, and `endcase`, 4 based on the control structure used.

For example:

```
<?php if ($Intval >= 20 &&  $Intval < 26): ?>
This text will only be displayed if the value of $Intval is more than or equal
to 20 and less than 26.
<?php endif; ?>
```

In the above example, the `if` statement is written using the alternative syntax and the HTML text is contained within it. The text would be displayed only if `$Intval` is more than or equal to `20` and less than `26`. You can use the syntax to similarly work with the other control structures and loops.

UNIT - 2
CONTROL STRUCTURES AND ARRAY

Chapter 4: Control Structures

Overview

Most of you have written programs at some time or another. The basic logic that you follow while writing a program, irrespective of the programming language, is to:

- Initialize variables.
- Evaluate a condition or make decisions.
- Print the output.

You already learned about declaring and initializing variables in Chapter 3. In this chapter, you will learn about evaluating conditions or making decisions. *Control structures* are the decision-making mechanisms of any programming language and are used for evaluating conditions. In any programming language, including PHP, control structures are of two types:

- **Conditional statements.** These statements execute a code if a condition is either True or False.
- **Conditional loops.** These loops run a set of instructions once or multiple times until a condition returns True or False.

By now you are familiar with the basics of PHP-variables, operators, and constants. In this chapter you will look at the various types of control structures that you can use in PHP. Control structures govern the flow of control in PHP scripts. Before you look at control statements and loops in detail, first look at what conditional expressions are.

Conditional Expressions

Decision making plays a major role in your life. You make decisions from the time you wake up to the time you go back to sleep! You perform actions based on the decisions that you make. Each decision involves evaluating a condition. If the condition evaluates to True, you perform certain actions; you perform another set of actions if the condition evaluates to False. For example, when you go shopping you might see a lovely vase that you always wanted to buy (I can definitely relate to this). The dilemma that you face or the decision that you need to make is, do you have money to buy the vase? If the answer is yes, then you can buy the vase. However, if the answer is no (oops!), then you cannot buy it. Similarly, *conditional expressions* are used in programming languages to evaluate conditions.

Conditional expressions consist of three components:

- **Operands.** These can be numeric, alphanumeric, and plain text. Plain text needs to be enclosed in double quotes (") before being evaluated.
- **Logical connectors.** These determine if all or only one of the conditions should return True for the code to execute.
- **Operators.** These are used to perform calculations and to compare expressions.

  Note To learn more about operators, refer to Chapter 3 , 'Variables, Operators, and Constants.'

Now look at some of the popularly used conditional statements available in PHP.

Conditional Statements

Sometimes you need to test a condition, and based on the result returned by this condition, you will perform certain actions. You can use conditional statements to do this. In a conditional statement, an expression is evaluated and, based on whether the condition returns True or False, the code is executed. In PHP, some of the commonly used conditional statements are:

- `if ... else ... else if`
- `switch ... case`

Both of these are covered in detail below.

The *if* Statement

The `if` statement is used to evaluate an expression and returns a Boolean value. The Boolean value can be either True or False. If the condition returns True, the set of statements following the `if` condition are executed. Otherwise the control is transferred to the statements at the end of the `if` block. Following is the syntax of the `if` statement:

```
if (expression)
   statement1;
```

In the above example, the condition is first evaluated and, provided the condition returns True, `statement1` is executed. Otherwise, nothing happens. However, you can also write the above example as:

```
if (expression)
        {
```

```
                    statement1;
                }
```

Note You might notice that the only difference between the first and second examples are the delimiters, or the curly braces ({}), as they are commonly known. By using delimiters, you can group together multiple statements and execute all of them together.

An example of the `if` statement is shown below:

```
<?php
 $Myvalue=5;
 if ($Myvalue ==5)
     {
             echo "I made a variable that contains the value 5";
             //The statement will execute only if $Myvalue is
equal to 5.
     }
?>
```

The above program assigns the value `5` to the variable `$Myvalue` and then checks to find if `$Myvalue` is equal to `5` . Since the condition returns True, the text is displayed on the screen.

```
<?php
  $Myvalue=5;
  if (!($Myvalue==4))
     {
             echo "See ! I told you ‘Myvalue' cannot be 4!";
             //This statement will always execute except when
$Myvalue is equal to 4.
     }
?>
```

In the above program, the value `5` is assigned to the variable `$Myvalue` and the text is displayed on the screen if `$Myvalue` is not equal to `4` . You can use the not (`!`) operator to specify that the statements should be executed only if the condition returns True. In the case of the (`!`) operator, if the expression returns True, then the result is False; if the expression is False, then the result is True. Consider another example:

```
<?php
  if (TRUE)
     {
             echo " I will always run."; //This statement will
always execute.
     }
?>
```

You can evaluate multiple conditions in the same `if` construct by using operators. Based on the operator used, you can execute a set of statements if all or one of the conditions is fulfilled. Following are some examples that use operators to evaluate multiple conditions. For detailed information on operators, refer to Chapter 3 , "Variables, Operators, and Constants ."

```
<?php
  $VarA=5;
  $VarB=6;
  if (($VarA==5) && ($VarB==6))
     {
```

```
                        echo "The text is shown only when the value of variable
‘VarA' is 5 and variable ‘VarB' is 6";
                    }
            ?>
```

In the preceding code, variables `$VarA` and `$VarB` are assigned values `5` and `6` , respectively. The condition is checked, and only if the condition returns True will the text be displayed on the screen. Here is another example where multiple conditions are evaluated simultaneously:

```
            <?php
              $Num1=5;
              $Num2=6;
              $Num3=7;
              $Num4=10;
              $Num5=12;
              $Num6=14;
              if ( ( (($Num1+$Num2)==11) && (($Num3+$Num4)==17)) ||
(($Num5+$Num6) < 50))
                    {
                            echo " OK!\n If this text is displayed, it means
that either sum of variable ‘ Num1' and ‘ Num2' is 11 and sum of variable
‘ Num3' and ‘ Num4' is 17 \n";
                            echo " OR,\n the sum of variable ‘Num5' and ‘Num6'
is less than 50.";
                    }
            ?>
```

In the above example, two main conditions are evaluated. The first condition contains two more subconditions. The first condition will return True if both of the subconditions return True. This means that the sum of `$Num1` and `$Num2` should be equal to `11` and the sum of `$Num3` and `$Num4` should be equal to `17` . The second condition will return True if the sum of `$Num5` and `$Num6` is less than `50` . If either of the two main conditions returns True, the statement contained in the `if` loop is executed.

> **Note** You can also nest one `if` statement within another to evaluate multiple conditions. This topic is covered later in the chapter.

Now consider how to enhance the use of the `if` statement by adding the `else` statement to it.

The *else* Statement

Consider a scenario where you want to execute a particular code segment when the condition evaluates to True. If the same condition (based on the `if` statement) evaluates to False, another set of statements must be executed. This is where the `else` statement proves to be useful. Certain points that you need to remember about the `else` statement are:

- The `else` statement can only be used along with the `if` statement and not separately.
- The code after the `else` statement will be executed only when the `if` statement returns False.
- You can have only a single `else` statement and it should always be the last set of code in the block.

The following example shows the use of the `else` statement together with the `if` statement.

```
            <?php
              $age=19;
```

```
            if ($age < 18)
                {
                        echo " You are not an adult since your age is less
than 18 ";
                        //This statement will not be printed.
                }
            else
                {
                        echo " You are an adult since your age is more than
18";
                        //This statement will be printed.
                 }
        ?>
```

In the above code, the value `18` is assigned to the variable `$age` and the program checks whether the value is less than `18` . Since the condition returns False, the code after the `else` statement is executed. This statement displays the text `"You are an adult since your age is more than 18"` .

Caution You cannot specify a condition along with the `else` statement. You can use the `else if` statement (covered below) to evaluate any additional conditions.

In an `if` conditional statement you can also evaluate other conditions, provided that the `if` condition returns False. To do so, you can use the `else if` statement.

The *else if* Statement

The `else if` statement is used along with the `if` and `else` statements. You can have multiple `else if` statements in a single `if` construct; however, there can be only a single `else` statement. An `else if` statement is executed only when the `if` condition and all the other `else if` statements return False.

Note You can use the words elseif and else if interchangeably because in PHP both mean the same thing and return the same value.

All the `else if` statements should be declared after declaring the `if` statement and before declaring the `else` statement in the same structure.

For example:

```
        <?php
          $Num=50;
           if (($Num>9) && ($Num<100))
               {
                        echo $Num, " is a 2 digit number";
                }
          elseif ($Num>=100)
                {
                        echo $Num, " has 3 digits or more !";
                }
          else
                {
                        echo $Num, " is a single digit number";
                }
        ?>
```

In the above code, `$Num` is assigned the value `50` and is checked against various conditions. If the value of `$Num` is

between 9 and 100 , a message "$Num is a 2 digit number" is displayed. If the value is more than 100 , the message states "$Num has 3 digits or more" . Otherwise the message reads "$Num is a single digi number" .

As stated earlier, you can have only a single else statement in an if structure. However, you can have multiple else if statements in the same if structure. This is illustrated in the following example.

```
<?php
  $Num=50;
  if (($Num>9) && ($Num<100))
      {
              echo  $ Num, " is a 2 digit number";
      }
  elseif (($Num>99) && ($Num<1000))
      {
              echo $ Num, " is a 3 digit number";
      }
  elseif (($Num>999) && ($Num<10000))
      {
              echo $ Num, " is a 4 digit number";
      }
  elseif ($Num>=10000)
      {
              echo $ Num, " is a 5 digit number or more";
       }
  else
      {
              echo $ Num, " is a single digit number";
      }
?>
```

The preceding example is similar to the one explained previously, except that two additional else if conditions are also evaluated. If the value of $Num is between 999 and 10000 , the message "$Num is a 4 digit number" is displayed. If the value is more than 10000 , the message states "$Num is a 5 digit number or more" . Otherwise the message states "$Num is a single digit number ".

Therefore, in case the if condition returns a False value, the variable is evaluated against each of the else if conditions. The moment one of the else if conditions returns True, the corresponding text is displayed and the control leaves the if structure.

The *break* and *continue* Statements

The break and continue statements are used in all the looping constructs. You use the break statement to end the execution of the current loop and return the control to the next statement immediately after the loop.

The continue statement does not move the control out of the loop; however, the rest of the statements in the loop after the continue statement are skipped and the loop is restarted.

Note You can also specify a numeric value along with the break statement, though this is optional. The specified number represents the number of loops that need to be skipped or ended. Just as in the break statement, you can also specify a numeric value along with the continue statement; this is also optional.

You have just learned about the if control structure, where a condition is first evaluated and code is executed based

on the result. Now look at how you can evaluate a condition and execute a code if the result matches one of the predefined values.

The *switch* Statement

In the case of the `if` construct, the condition is either evaluated by using the `if` statement or the `else if` statements. However, in the case of the `switch` construct, the condition is evaluated only once and then the value is compared against a list of predefined values. Each value is stored in a `case` statement and refers to the same variable. When a matching value is found, the control shifts to the specific `case` statement and all the statements under it are executed sequentially. If no matching value is found, then the statements stored under the `default` label are executed until the end of the block is reached or you come across a `break` statement. In case neither a matching value is found nor a `default` label exists, the control shifts to the statements outside the `switch` block.

Following is the syntax of the `switch` statement.

```
switch (variable)
   {
         case value1:
                                statement;
                                statement;
         case value2:
                                statement;
                                statement;
           ...
         default:
                                statement;
                                statement;
   }
```

Note Not every `case` statement needs to contain a value. You can leave a `case` statement empty. This means that you can omit adding statements in the body of the `case` structure. This is helpful in cases where you do not want the program to perform any action if the variable matches a certain `case` .

Expressions specified in a `case` statement can only belong to the `integer` , `string` , or `float` datatype. You cannot use `arrays` or `object` values as `case` expressions.

Following are some examples of the `switch` statement.

```
?php
  $val="first";
  switch ($val)
  {
          case "first":
             echo "You have encountered the first case.", "\n";
          case "second":
             echo "You have encountered the second case.", "\n";
          case "third":
             echo "You have encountered the third case.", "\n";
          case "fourth":
             echo "You have encountered the fourth case.", "\n";
             break;
  }
?>
```

The output of the above code is shown in Figure 4-1 .

```
X-Powered-By: PHP/4.0.4pl1
Content-type: text/html

You have encountered the first case.
You have encountered the second case.
You have encountered the third case.
You have encountered the fourth case.
```

Figure 4-1: The `switch case` construct without `break` in each statement

In the above example, if the value of the variable `$val` is `fourth` , the last statement will be executed. However, if the value is `first` , then all the statements in each of the `case` statements are displayed. To avoid this, you should insert a `break` statement at the end of each `case` statement. The `break` statement forces the control out of the `switch` construct to the next statement immediately after the loop. You can rewrite the above code as follows.

```
?php
  $val="first";
  switch ($val)
  {
          case "first":
             echo "You have encountered the first case.", "\n";
             break;
          case "second":
             echo "You have encountered the second case.", "\n";
             break;
          case "third":
             echo "You have encountered the third case.", "\n";
             break;
          case "fourth":
             echo "You have encountered the fourth case.", "\n";
             break;
  }
?>
```

The output of the above code is shown in Figure 4-2 .

```
X-Powered-By: PHP/4.0.4pl1
Content-type: text/html

You have encountered the first case.
```

Figure 4-2: The `switch case` construct with `break` in each statement

Now that you know about control statements, it's time to look at the various looping constructs that are available in PHP.

Conditional Loops

You use conditional loops to execute a set of statements until a condition remains True. To ensure that the loop performs correctly, the value that is evaluated in the condition should be modified every time the loop is executed.

Some of the common conditional loops that are available in PHP are

- `for`

- `while`
- `while ... endwhile`
- `do ... while`
- `foreach`

Each of these conditional loops will be discussed in detail below. The first looping construct that you will look at is the `for` loop.

The *for* Loop

`For` loops are loops in which the number of times the loop will be executed is fixed and the information is included in the statement itself. A `for` loop evaluates three different expressions, which are also the various stages of the loop.

- **Initialization of variables.** You assign a value to the variable in `expr1` . The assigned value is generally the starting point of the loop.
- **Evaluation of the condition.** The conditional expression (`expr2`) is evaluated at the beginning of each loop. If the condition evaluates to True, the loop continues and the statements inside the loop are executed. However, if the condition evaluates to False, the execution of the loop ends and the program exits the loop.
- **Reinitialization of the variables once the code executes.** At the end of each loop, `expr3` is evaluated, or rather executed. Mostly, `expr3` is not a conditional expression and mainly involves incrementing the value of the variables used in the code.

The syntax of the `for` loop is as follows:

```
for (expr1;expr2;expr3) statement;
```

Since all the instructions are in the same line of code, it is easier to follow the flow of control and understand the execution of the code in the loop.

You can have a single statement inside a `for` loop. However, it is preferable to have the `statement` on a separate line. This makes the code more flexible. You can also write the above `for` loop as shown below.

```
for(expr1;expr2;expr3)
   {
   statement1;
   statement2;
   }
```

The second syntax is preferable because it is more flexible and you can execute multiple statements at one go. When the code is executed, all the statements written between the braces are performed before incrementing the variable and reevaluating the condition. Consider a few examples of the `for` loop.

```
<?php
 for ($Myval = 1; $Myval <= 10; $Myval++)
      {
              echo $Myval , "\n";
      }
?>
```

It is not mandatory to have values in all the expressions in the `for` loop. You can leave any, or all, of the expressions in a `for` loop blank. This makes the `for` statement very flexible and one of the most popular looping constructs in PHP.

For example:

```
<?php
 echo " LOOP 1 \n";
 $Myval=1;
 for ( ;$Myval <= 10; $Myval ++)
      {
               echo $Myval;
      }
 echo "\ n                LOOP 2 \n";
 $Myval=1;
 for ( ;$Myval <= 10;)
       {
                echo $Myval;
                $Myval ++;
         }
 echo "\n                LOOP 3 \n";
 $Myval =1;
 for (;;)
         {
            if ($Myval >10) {break;}
                echo $Myval;
                $Myval ++;
         }
 echo "\n";
?>
```

The output of the preceding code is shown in Figure 4-3 .

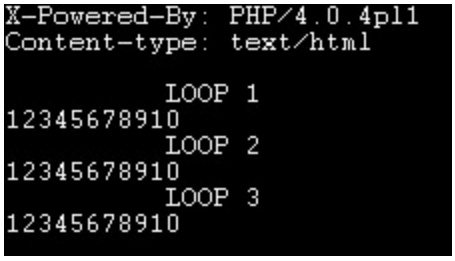

Figure 4-3: Three `for` loops with and without parameters

As you might have already noticed, the first loop has only two parameters, the second loop has only one, and the final loop does not have any parameters. Despite this, all three loops will produce the same output. Every programmer can modify the loops based on their requirements.

The `for` loop executes a code for a predefined number of instances. Suppose that you do not know how many times the code would be executed or whether you need to execute a code until a condition return False. In such cases, you can use the `while` loop.

The *while* Loop

You use the `while` loop to check whether an expression is True or False. All the statements inside the `while` loop will execute repeatedly, as long as the expression remains True. On the other hand, the loop might not execute at all This is because the `while` loop checks the expression before entering the loop. This means that if the condition is False, the compiler will not enter the `while` loop even once. The syntax of the `while` loop is as follows:

```
while (exprn) statement;
              or
while (exprn)
  {
         statement1;
          statement2;
  }
```

Following are a few examples of the `while` loop.

```
<?php
 $Num=5;
 while ($Num)
  {
                echo "I will print infinitely. Press Ctrl +
C to stop me. \n";
                  //This is because as long as $Num returns
the value 5 the loop will go on.
      }
?>
<?php
   $Num=5;
   while ($Num<10)
      {
                echo "I will also print infinitely. Press
Ctrl + C to stop me.\n";
                //This is because $Num will always return 5
       }
?>
<?php
  $Num=5;
    while ($Num<10)
        {
    echo "I will print till variable ‘Num' is less than
10. Right now the value of Num is ", $Num, "\n";
                ++$Num;
     }
                echo " OK! We are out of the loop and the
current value of the variable ‘Num' is ", $Num;
?>
```

The `while` loop also supports all valid expressions, including those with operators. You use the && (And) or the || (Or) operators to check for two conditions. In the case of the && operator, both the conditions should return True. The loop will end if either of the conditions returns False. In the case of the || operator, the code is executed if either of the conditions returns True.

For example:

```
<?php
  $Num1=1;
  $Num2=35;
  echo "The loop stops executing if variable ‘Num1' is 15 or greater
and if variable ‘Num2' is 18 or
  lesser";
```

```
            while (($Num1<15) && ($Num2>18))
                {
                        ++$Num1;
                        --$Num2;
                        echo " Variable ‘Num1' = ", $Num1," and variable
‘Num2'= ", $Num2 ,"\n";
                        //The loop will end even if one of the two sub-expressions
returns False.
                }
                echo " We are out of the loop and, ‘Num1' = ", $Num1," and ‘Num2'=
", $Num2 ,"\n";
            ?>
```

In the above example, the `while` loop continues until either `$Num1` becomes more than or equal to 15 or `$Num2` becomes less than or equal to 18. Now that you know

about the `while` loop, take a look at another kind of looping construct, the `while ... end while` loop.

The *while ... end while* Loop

The `while ... endwhile` loop is very similar to the `while` loop. The only difference is that in the `while` loop you use delimiters to mark the loop segment. However, in the `while ... endwhile` loop, all statements after the `while (expr )` statement and before the `endwhile` statement are considered inside the loop and executed sequentially. The syntax of the `while ... endwhile` loop is as shown below.

```
while (expr):
        statement1;
        statement2;
endwhile;
```

For example:

```
<?php
  $Num = 1;
   while ($Num <= 10):
              echo $Num;
              $Num++;
    endwhile;
?>
```

The output of the above code will appear as shown in Figure 4-4 .

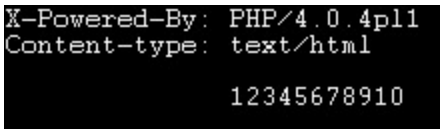

Figure 4-4: The `while` loop

In the above example, the `while` loop will execute as long as the value of `$Num` is less than or equal to 10. The moment the value becomes greater than 10, the control exits the loop and is transferred to the statements immediately after the end of the `while` loop. The third loop in the `while` loop series is the `do ... while` loop, in which the code is executed at least once.

The *do ... while* Loop

The `do ... while` loop is also similar to the `while` loop. However, in the case of the `do ... while` loop, the code executes once before the condition is evaluated. All the statements after `do` and before `while (expr)` are considered as a part of the loop, and you need to enclose all the statements in delimiters. The syntax of the `do ... while` loop is as shown below.

```
do
      {
         statement1;
         statement2;
      }
while (expr)
```

As you might have noticed by now, in the `do ... while` loop the expression is evaluated at the end of the loop. This means that even if the condition is False the code in the loop is still executed at least once.

For example:

```
<?php
  $Num=1;
  do
      {
       echo " Let us increase the value of variable ‘Num' to get
out of this loop! variable is $Num \n";
                ++$Num;
      }
   while ($Num <20);
  echo "Hurray ! I am out of the trap! Variable ‘Num' =", $Num;
?>
```

The above code will run 20 times, since the initial value of `$Num` is `1` . However, if you initialized `$Num` with the value `21` , the code would still be executed once, even though the condition is False. This is because the code is executed in the beginning of the loop and the condition is only evaluated at the end of the loop.

Note Some of the common syntax errors found in loops can be avoided by remembering to put the semicolon in the `while` and the `do ... while` statements.

The `while` and `do ... while` loops execute a code if a condition evaluates to True, but suppose you need to execute a single code for each element of an array or a group of objects. You can do this by using the `foreach` loop.

The *`foreach`* Loop

In the case of the `foreach` loop, a set of code is executed against a set of elements of an array or object. Each element of the array is evaluated independently and the control leaves the loop only after all the elements of the array have been evaluated. The control passes to the next statement after the end of the `foreach` loop. The syntax of the `foreach` loop is as follows:

```
foreach(arrexpression as $currval) statement
```

or

```
foreach(arrexpression as $sourceval => $currval) statement
```

In the first syntax, all elements in the array are traversed and the variable is incremented before the next loop begins In the second syntax, the current value of the variable is first stored in another variable `$sourceval` before incrementing `$currval` .

You can also use the `for` loop to find out the number of elements in an array or the number of objects in a collection

Note You use the `break` and `continue` statements to exit the loop or to skip statements in the loops.

Consider some examples of the `foreach` loop.

Example 1:

```
<?php
$arrayval = array (12, 22, 32, 72);
foreach ($arrayval as $currval)
      {
                  echo "The current value stored in the array \$
arrayval: $currval \n";
      }
?>
```

Figure 4-5 shows the output of the above code.

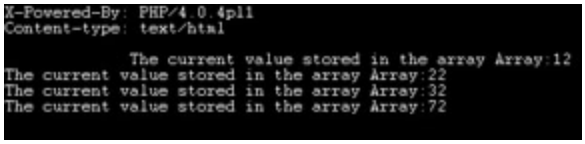

Figure 4-5: The `foreach` loop displaying each value of an array

Example 2:

```
<?php
$arrayval = array (Jack, Tom, Mary, James);
$count = 0;
foreach($arrayval as $currval)
      {
                  echo "\n $arrayval [$count] => $currval.\n";
      }
?>
```

Figure 4-6 shows the output of the above code.

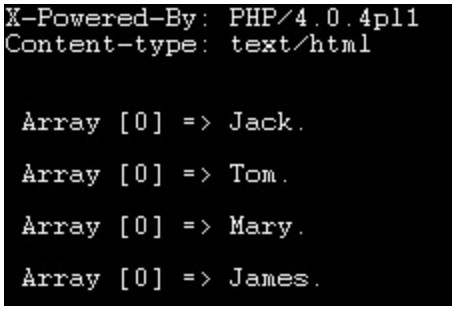

Figure 4-6: The `foreach` loop displaying each value of an array

Example 3:

```
<?php
```

```
$arrayval = array (
            "Sunday          "    => 1,
            "Monday         "    => 2,
            "Tuesday        "    => 3,
            "Wednesday    "    => 4,
            "Thursday       "    => 5,
            "Friday           "    => 6,
            "Saturday       "     => 7
    );
    foreach($arrayval as $tempval => $currval)
    {
            echo "\$arrayval[$tempval] => $currval.\n";
    }
?>
```

Figure 4-7 shows the output of the above code.

Figure 4-7: The `foreach` loop displaying each value of an array

In all the three examples, each element of the `$arrayval` array is displayed separately. Once the current value is displayed, the value of the counter is incremented and the next element is displayed.

Until now you have only worked with a single level of control structure, but you can also have multiple levels of control structures.

Nested Control Structures

You have already learned how to use each control structure separately. You can also insert or nest one control structure inside another. For example, in a `for` loop you can evaluate one condition and run another `for` loop inside the first loop to evaluate another condition. These are known as *nested control structures* .

You can nest control structures to as many levels as you require. As good programming practice, you should indent each level of the structure for better readability and understanding.

For example:

Example 1:

```
<?php
  for ($Num1=1;$Num1<5;$Num1++)
      {
              for ($Num2=1;$Num2<5;$Num2++)
                    {
                              echo" I will be printed 25 times
total \n";
                    }
                    }
?>
```

Figure 4-8 shows the output of the above code.

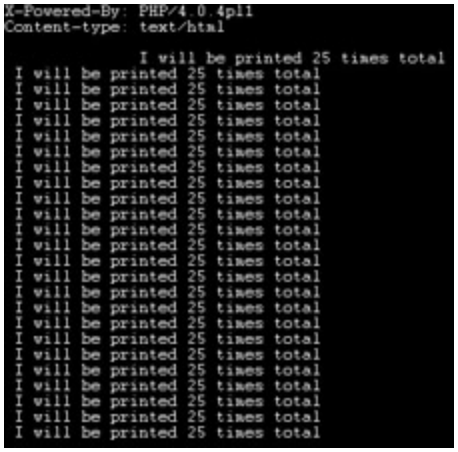

Figure 4-8: Nested control structure using `for` loops

In the above code the outer `for` loop is first evaluated and is executed until `$Num1` is less than 5. For each instance of the outer loop, the inner `for` loop runs five times. Therefore, in total the text is displayed 25 times. See Figure 4-8 .

`Example 2:`

```
<?php
$arrayval [0][0] = "2";
$arrayval [0][1] = "6";
$arrayval [1][0] = "8";
$arrayval [1][1] = "12";
  foreach($arrayval as $currval1)
    {
      foreach ($currval1 as $currval2)
          {
              echo "$currval2 \n";
          }
    }
?>
```

In the above code, the inner loop is executed for a specific number of instances depending upon the number of elements in the array. For each instance, the current value of `$currval2` is displayed and the counter incremented.

Chapter 5: Arrays

Consider that you need to store 100 values. To do so you must define 100 variables and assign a value to each variable. However, the task of defining 100 variables is time-consuming and tedious. Moreover, it is difficult to juggle so many variables at the same time. To solve the difficulties that arise due to handling many variables, you can define an array that can store 100 values. Besides storing one or more values simultaneously, an array is also flexible and comparatively easy to handle. You'll find that the manipulation of arrays and their elements is much simpler and faster than manipulation of many variables at any given time.

In the previous chapter you learned about variables and constants, the building blocks of the PHP language. In this chapter you will learn to work with arrays, a fixture in most of the programming languages. You will learn about the different types of arrays that are available in PHP. You will also learn to create and manipulate these arrays.

Introduction to Arrays

You already know that a variable is a container for a single value. An array, on the other hand, is a container for multiple values. An array can also contain multiple elements having different values. PHP also allows an array to contain elements of different data types. Elements in an array are referred or manipulated on the basis of their *index*. The index of an element in an array is normally an integer.

> **Note** Indices in PHP are also referred to as *keys*.

There are many types of arrays available to you in PHP. Consider the following types of arrays in detail.

Types of Arrays

Arrays can be classified on the basis of indices and array elements. The different types of arrays include:

- Enumerated arrays
- Associative arrays
- Multidimensional arrays

Enumerated Arrays

Arrays that have integer indices are known as *enumerated* or *numerically indexed* arrays. These arrays are used to store values. For example, you can have an array named `Students`, where each element of the array stores the name of a student.

In the above array, each individual element can be referred to by its index number. For example, the name James can be referred to by the index `0`. To display the array element at index `2`, you can use the following statement:

```
echo "$Student[2]";
```

> **Note** By default, the index number of an array element starts with zero and increases sequentially. Therefore, the index of the last element in an array is always one less than the total number of elements in that array.

Associative Arrays

If you need to access elements in an array by name, then you should use strings as an index of the array. You can use a string as an index of an array element to store both values and names. The arrays that use strings, as indices, are referred to as *associative arrays*.

To access an array

You can use the command `Print "Student [x]"` instead of the `echo("Student [x]")` command to access the value of the array elements according to their index numbers. Here *x* refers to the index number of the element in the array named `Student`.

You will find one subtle difference while working with numerially indexed (enumerated) arrays and string indexed (associative) arrays. With associative arrays having strings as an index, you will not be able to compute the next valid index in the array. However, indices can be assigned to the `Student` array discussed earlier using the array identifier. You can create an associative array `Student_Name` by assigning a value to the index.

```
$Student_Name["name1"] = "James Patt";
$Student_Name["name2"] = "John Smith";
$Student_Name["name3"] = "Susan Carter";
```

You can also convert an existing enumerated array into an associative array. To convert the existing `Student` array into an associative array:

```
$Student = array ( "name1" =>"James Patt", "name2" =>"John Smith",
"name3" =>"Susan Carter");
```

After you have converted an enumerated array into an associative array, you can use the `echo` command to access the elements of the given array.

```
echo "$Student[name3]";
```

You will get the output as `Susan Carter`.

You can also use `list()` and `each()` functions to access the elements in string indexed arrays. The `list()` function is used to assign the values of array elements to variables. On the other hand, the `each()` function takes an array as the argument and returns the index and value of each element of the array.

Note You'll learn about functions in detail in Chapter 6, 'Functions.'

For example:

```
$Student = array ("name1" =>"James Patt", "name2" =>"John Smith",
"name3" =>"Susan Carter");
list($key_name, $val) = each($Student);
echo("$key_name");
echo("$val");
```

The preceding code snippet sets `$key_name` equal to the index of the element and `$value` equal to the value of the element for each element of the array. Next you display the value of `$key_name` and `$val` using the `echo()` command. Therefore, the indices will be `name1`, `name2`, and `name3`. The corresponding values would be `James Patt`, `John Smith`, and `Susan Carter`.

Multidimensional Arrays

You can store different variables as well as complete arrays in another array. An array that stores arrays as its elements is known as a *multidimensional array*. Again, if the array elements in the two-dimensional array contain arrays, you have a three-dimensional array, and so on. Although these arrays can be three-dimensional, four-dimensional, etc., two-dimensional arrays are the most popular. An illustration of a multidimensional array is given below.

```
Roll No     1             2             3
Name        James Patt     John Smith     Susan Carter
```

The array illustrated above is an example of two-dimensional arrays. Here the `Student` array contains another array called `Roll No` within it.

Consider that the `Student` array contains arrays as elements. Therefore, if you want to access the second sub-element of the first element of the `Student` array, you have to use two indices as shown below:

```
$Student[0][1]
```

You can also think of an image as an example of an array. For instance, you need to trap the coordinates of an image as you move the mouse cursor over it. In this case, you need to use a multidimensional array to trap the x and y coordinates as shown below:

```
Coordinates[x][y]
```

There is no limit on the number of dimensions that you can have for the arrays. There can also be a combination of dimensions. For example, you can have the first dimension in your array indexed by integers, the second dimension indexed by strings, the third dimension indexed by integers, etc. Multidimensional arrays are very useful in representing complex statistical data.

```
 <html>
<body>
<?php
$Student = array (
                      "0"=> array ("name"=>"James", "sex"=>"Male", "age"=>"28"),
                      "1"=> array ("name"=>"John", "sex"=>"Male", "age"=>"25"),
                      "2"=> array ("name"=>"Susan", "sex"=>"Female", "age"=>"24")
                       );
Print $Student [2][age];
?>
</body>
</html>
```

The output of the code is `24`.

Now that you have learned to identify an array, you will next learn to create and initialize an array in PHP.

Initializing Arrays

An array can be initialized in two ways. You can initialize an array using:

- An array identifier
- The `array()` function

Using the Array Identifier

The *array identifier* is an empty set of square brackets. You can use an array identifier to initialize the `Student` array as shown below:

```
$Student[] = "James";
$Student[] = "John";
```

Here you assigned values `"James"` and `"John"` to the `Student` array. Note that you have not specified the indices for the array. Therefore, indices `0` and `1` have been assigned respectively to James and John.

You can also specify the indices for an array explicitly. For example:

```
$Student[0] = " James";
$Student[1] = " John";
```

Normally, indices are assigned sequentially in arrays. But you can also assign indices to array elements randomly. For example:

```
$Student[10] = " James";
$Student[3] = " John";
```

In the above example, you assigned indices to arrays in a nonsequential manner. If you need to assign indices to the elements of the array, the next index value will begin after the highest-indexed element in the array. Consider that you want to assign an index to the array element containing the name `Sarah`. Here the highest index among the elements is 10. Therefore, the element containing the name Sarah will have an index `11`.

Using the `array()` Function

A simpler way to initialize an array is by using the `array()` function. This is simpler because, by using the `array()` function, you can assign multiple values to an array simultaneously. You can use the `array()` function to define the array `Student` in the following manner:

```
$Student = array("James", "John", "Susan");
```

> **Note** You normally use the default indices while working with arrays. This implies that the indexing of an array would normally start with `0`. However, you can also override the default indices using the => operator. In the above example, the `Student` array has three elements with indices `0`, `1`, and `2`. However, you can specify the indices to start from `1`. To do so, you can write the following code:
> `$Student = array (1 => "James", "John", "Susan");`

Now if you use the `Print "Student [1]"` command, you will get the output as `James`.

You can also use the => operator in the following manner:

```
$Student = array ("James", 5 => "John", "Susan");
```

In the preceding example, you have assigned an index of `0` to James, `5` to John, and `6` to Susan.

So far, you have learned to identify the different types of arrays. You have also learned to initialize arrays in your PHP codes. Next you will learn to manipulate the arrays using different functions that are supported by PHP.

Working with Arrays

Using the built-in functions, you can perform the following tasks:

- Modify the size of an array.
- Loop through the array.
- Find elements in the array.
- Reverse an array.
- Sort an array.

The following sections will discuss the functions in detail.

Modifying the Size of an Array

You can modify the size of an array by using the different functions supported by PHP. By using these functions, you can:

- **Determine the size or number of elements of an array.** You can access an element of an array using the index. However, to calculate the number of elements contained in an array, you need to use the `count()` function. You can also calculate the index number of the last element of the array. To understand the `count()` function better, look at the following code snippet.

  ```
  $Student = array ( "Susan", "Betty", "Reggie" );
  $Number = count ($Student);
  Print $Number;
  ```

  The above code initializes the array `Student` with three elements. The `count()` function returns a count of the number of elements in this array. This count is stored in a variable called `Number`. When you display the value of this variable, the output is `3`. To find out the index number of the last element in the array, you need to subtract `1` from the variable `Number`. For example:

  ```
  Print "$Student[count($Student) -1]";
  ```

- **Change the size of an array.** You can decrease or increase the length of an array using different functions in PHP. Changing the size of an array involves:
  - **Reducing the size of an array.** To reduce the size of an array, you use the `array_slice()` function. The `array_slice` function accepts an array, a starting position, and length as the parameters. For example, you have an array that consists of seven elements. To reduce the number of elements from seven to four, use the `array_slice` function.

    ```
    $Student = array( "ab", "bc", "cd", "de", "ef", "fg", "gh");
    array_slice($Student, 4)
    ```

- The `array_slice()` function reduces the number of elements from seven to four.

  You have learned to reduce the length of an array. However, this reduction in the number of elements of an array takes place from the end. If you wish to remove the first element of an array, you will need to use

the function `array_shift()`.

The `array_shift()` function removes the first element of an array. The elements are passed to the array as arguments. For example, you initialize:

```
$Student = array("James", "Susan", "John");
```

Now, you can use the `array_shift()` function and store the output in a variable called `New` as:

```
$New = array_shift($Student);
```

The first element of the `Student` array `"James"` would be automatically removed by the `array_shift()` function.

Note You can also delete an element from an associative array permanently by using the `unset()` function. The `unset()` function is a built-in function in PHP. Using the `unset()` function, you can destroy more than one variable at a time. You can also delete index values from an associative array.

- **Increasing the length of an array.** Assume that you need to insert some elements to the `Student` array mentioned earlier. You can do so by using the `array_push()` function. The `array_push()` function accepts an array as a parameter. The `array_push()` function returns the total number of elements in the array. Consider the following code snippet:

```
$Student = array("Susan", "James", "John");
$Insert = array_push($Student, a, b, c);
```

Here you have taken the `Student` array and added elements `a`, `b`, and `c` to it. You then store the total number of elements into another array called `Insert`. Therefore, the `Insert` array has six elements.

Note If you want to add a new element into an associative array, you need to specify the index and value for the array and add it. For example,

```
$Array_1 =array("a" =>"1", "b" =>"2", "c" =>"3");
$Array_1 =array("d" => "4");
```

In the above code, you have added a new element to `Array_1`.

Now suppose that you want to add all the elements of an array to another array. In that case, you should add arrays instead of adding individual elements because it is faster and easier. You will not be able to add one array to another with the `array_push()` function. In PHP, there is a function called `array_merge()` that allows you to merge one or more arrays.

The `array_merge()` function merges two or more arrays and returns a combined array.

For example, consider the following:

```
$Student= array("John", "James", "Susan");
$Class = array("4","5","6");
$Combine = $Student + $Class;
```

The `Combine` array contains the copies of all the elements of both `Student` and `Class` arrays. Therefore, the `Combine` array will contain the following values:

```
"John", "James", "Susan", "4", "5", "6"
```

Now suppose that the arrays `Student` and `Class` contain some common elements between them. For example:

```
$Student= array("John", "James", "5");
$Class = array("4","5","6");
```

Now when you merge the two arrays, the common element is copied only once into the merged array. However, if you don't want to lose the common element in the two arrays, you can use the `array_merge_recursive()` function. You can also use this function to avoid losing indices when merging arrays. The `array_merge_recursive()` function copies all the elements that are in the arrays.

- **Computing intersection or union of arrays.** Sometimes you might need to determine the intersection of two arrays. In that case, you should use the `array_intersect()` function. For example, look at the following snippet:

```
$old = array ("a", "b", "c");
$new = array ("c", "d", "e");
$Intersection = array_intersect ($old, $new);
```

In the preceding code snippet, first you initialize two arrays-`old` and `new`. Then you find the intersection of the two arrays and store the result in another array.

Note If you wish to find similarities between two arrays, first you should use this function to find the intersection. Then you should use the `count()` function to count the number of elements. The `count()` function takes the following form.

```
$Count_intersection = count($Intersection);
```

In the above statement, `Count_intersection` is the variable used to store the number of elements after intersection.

If you wish to determine all the elements of both arrays, use the `union` function. The `union` function takes the following syntax:

```
Union = (array);
```

- **Extracting unique elements from an array.** In order to extract a unique element from an array, you need to use the `array_unique()` function. This is a built-in function in PHP. The `array_unique()` function passes the array as argument, removes the duplicate elements, and returns a new array. You can use this function on both numerically indexed and associative arrays. You use the `array_unique()` function in the following manner:

```
$unique = array_unique ($old);
```

You learned to modify the size of an array, find the intersection and union of two arrays, and determine unique elements within an array. Now you will see how to navigate through arrays.

Looping through an Array

To perform operations on the elements of arrays repeatedly, you need to navigate through the arrays in a continuous manner. There are different ways of looping through each element of an array. However, the most powerful method is using the `foreach` statement of PHP. Consider an example.

You have a numerically indexed array called `Student` that contains `"John"`, `"James"`, and `"Susan"` as array elements. Now you use the `foreach` statement to access each element at a time and store each element temporarily in a variable called `new`. The syntax of the command is as below:

```
$Student = array("John", "James", "Susan");
foreach ($Student as $new)
               {
                print "$new";
```

```
}
```

The above-mentioned code snippet will display all the array elements. You used the `foreach` statement to loop through a numerically indexed array. To loop through an associative array and access its indices and values, you need to modify the `foreach` statement in the following manner:

```
foreach($Student as key_name=>$new)
          {             //
          }
```

Here `key_name` is the variable that stores the index and `new` is the variable that stores each value temporarily.

> **Note** To loop through sequentially indexed arrays, you can use the `count()` function to find out the number of elements in the array and then use a `for()` loop.
>
> Again, to loop through nonsequentially indexed arrays, you can use the `current()` function and the `key()` function. The `current()` function is used to determine the value of the current element in the array and the `key()` function is used to determine the index of the current element.

You learned to navigate through an array using the `foreach` statement. You also learned to use the `count()` and `current()` functions. You can use two other functions to navigate through arrays in PHP. These two functions are the `next()` and the `prev()` functions.

The `next()` function takes an array as its argument. This function traverses from the left to right of an array. The `next()` function returns the value of the next element. Whenever it reaches the last element, it returns a value of False. The `prev()` function also takes an array as its argument and traverses from the end to the beginning of the array.

> **Note** Assume that you have written a function `f1()` that displays a text. You want this function to be executed automatically whenever you access any element of the `Student` array. For it to do so, you have to navigate through the array and apply the `f1()` function to all the elements of the array. You can use the `array_walk()` function for this purpose. The `array_walk()` function takes two arguments. The first argument is the array and the second argument is the name of the function to be applied. The `array_walk()` function takes the following form:
> `array_walk ($Student, f1);`

You have learned to navigate through arrays. Now you will learn to find elements in arrays.

Finding Elements in an Array

PHP provides some functions to find elements from arrays. In PHP, you will be able to:

- **Find elements that match a certain criteria.** Consider that you need to store those elements from an array that match a certain criteria. You can find these elements in two ways. You can use the `foreach()` construct to navigate through the array and test the value of each element in the array. You can also use the `preg_grep()` function to perform the same task. The `preg_grep()` function searches the array for the given criteria and returns the elements matching the criteria as an array.

  Consider that you need to search the `Student` array and list all the students having Smith as their surname. The usage of the `preg_grep()` function is shown as follows:

  ```
  $Student = array ("James Patt", "John Smith","
  Susan Carter", "Joe Smith", "Janet Jones");
  $array_Smith = preg_grep ("/^Smith", $Student);
  ```

In the code snippet written above, you have searched the array `Student` with the `preg_grep()` function for the string `Smith`. The above code will give `John Smith` and `Joe Smith` as the output.

Note You use the `preg_grep()` function along with the `array_shift()` method to find the first element in an array that matches a given criteria. You can also use the `while` loop to find the first relevant match. The `preg_grep()` function is very useful in searching short arrays efficiently. However, if the array is large, using the `preg_grep()` function will result in very slow processing.

- **Find elements in one array but not another.** Till now, you have determined unique elements in an array. Now suppose you need to determine those elements in an array that are not present in another array. You can use the `array_diff()` function to find the difference between the two arrays. The `array_diff()` function takes the following form:

  ```
  array_diff ($Array_1, $Array_2);
  ```

  In the above example, you have used the `array_diff()` function to determine the elements present in `Array_1` and not in `Array_2`. You can use this function to search more than one array. You use commas to specify more than two arrays. This function usually allows duplicate elements. To eliminate the duplicate elements, you need to use the `array_unique()` function.

Now that you have learned to find the difference between two arrays, it's time to move on to reversing arrays.

Reversing an Array

Consider that you need to process an array in the reverse order. If the array is short, you can use the `array_reverse()` function provided by PHP. The `array_reverse()` function takes one argument, the array. The `array_reverse()` function takes the following form:

```
$array2 = array_reverse ($array1);
```

If you have an array that is large, the `array_reverse()` function will prove to be time-consuming. To reverse an array that is big in size, you can use a `for()` loop to process the array backward. For example, you can reverse the `Rate` array (discussed earlier) using the `for()` loop as shown below:

```
for (x = count ($Rate) -1; Rate >=0; x--)
{
//code
}
```

Here you first find out the total number of elements by using the `count()` function. Next you move to the index of the last element of the array. Finally, you navigate through the array in the reverse order.

You have learned to add or remove elements from an array. You also have learned to find elements from arrays. Now you will learn to sort the arrays with some functions provided by PHP.

Sorting and Unsorting Arrays

The `sort()` function is a simple function used for sorting arrays.

The *`sort()`* Function

You can use the `sort()` function to sort arrays according to the numeric and alphabetical order. This function also changes the indices of the arrays according to the sorted order. For example:

```
$Student = array ("John", "James", "Susan");
```

```
Sort ($Student);
```

When you display the index number and the value of the array elements, you get the output as:

```
0          James
1          John
2          Susan
```

Note that the indices have also changed after sorting the array.

Now assume that you have an associative array. For example:

```
$Student = array ("JS" -=> "John Smith",
                        ("JP" -=> "James Patt","
                        ("SC" -=> Susan Carter");
```

If you sort the array using the `sort()` function and then display the index number and value of the array element, you get the output:

```
0          James
1          John
2          Susan
```

Note that the string indices of the associative array have been changed to numeric indices. This is the problem when you sort an array using the `sort()` function. This problem can be avoided if you sort the arrays using the `asort()` function. The `asort()` function will be discussed next.

There are other functions that help you to sort associative and numerically indexed arrays. These are:

- `asort()`: The `asort()` function is used to sort arrays without changing the indices. If you sort the `Student` array mentioned earlier using the `asort()` function, you will get the following output:

  ```
  JP           James Patt
  JS           John Smith
  SC           Susan Carter
  ```

- `rsort()`: The `rsort()` function is similar to the `sort()` function. The only difference is that it sorts the arrays in the reverse order. Consider the following code:

  ```
  $Student = array ("John", "James", "Susan");
  rsort ($Student);
  ```

  When you display the index number and the value of the array elements after sorting the array using the `rsort()` function, you get the following output:

  ```
  0           Susan
  1           John
  2           James
  ```

- **`arsort()`:** The `arsort()` function is similar to the `asort()` function. The only difference is that it sorts the arrays in the reverse order. Assume that you need to sort the `Student` array in reverse order such that the indices are not modified. To sort an array in the reverse order without modifying the indices, you need to use the `arsort()` function.

  Using the `arsort()` function to sort the `Student` array, you would get the following output:

  ```
  SC           Susan Carter
  JS           John Smith
  JP           James Patt
  ```

- **ksort():** To sort the indices of an associative array, you need to use the `ksort()` function. Assume that you have an array `Student` and you need to sort according to the indices.

  ```
  $Student = array ("b" =>"Susan", "d" =>"Andrew",
  "a" =>"John", "c" =>"James");
  ```

  Sorting the `Student` array using the `ksort()` function will provide the following output:

  ```
  a            John
  b            Susan
  c            James
  d            Andrew
  ```

  Therefore, the `Student` array has been sorted according to the indices.

- **ukrsort():** The `krsort()` function is similar to the `ksort()` function. The only difference is that this function sorts the arrays in reverse order. Therefore, if you use `krsort()` in the example of the `Student` array mentioned above, you will notice the following output:

  ```
  d            Andrew
  c            James
  b            Susan
  a            John
  ```

- **usort():** The regular sorting functions of PHP do not support sorting multidimensional arrays. To sort multidimensional arrays, you need to use the `usort()` function. You can also use the `usort()` function to specify a user-defined function for sorting the array. The `usort()` function takes an array and a function as the arguments. PHP provides two functions to sort associative arrays using user-defined functions. You can use the `uasort()` function to sort an associative array. You can also use the `uksort()` function to sort an array by the indices.

You have learned to sort the arrays. Now you will learn to randomize an array. You can randomize an array using the `shuffle()` function.

shuffle()

You use the `shuffle()` function to randomize or unsort arrays. The `shuffle()` function changes the order of the elements in the array. The `shuffle()` function takes the following form:

```
$Array_shuffle = range(1,100);
shuffle($Array_shuffle);
```

Here you use another function, known as the `range()` function. The `range()` function takes two integer parameters, the start and the end point. This function returns an array of all the integers between 1 and 100. The `shuffle()` function then randomizes the integers from 1 to 100 and stores them in the array called `Array_shuffle`.

Chapter 6: Functions

Overview

Suppose that during the festive season an online bookstore is offering a 10 percent discount on every purchase that costs $90 or less. For any purchase priced above $90, the discount rate is 12 percent. So every time a book is sold, the discount needs to be calculated on the basis of the total purchase. The discount is then deducted from the net price to be paid by the customer. Now suppose that on a particular day 23 customers placed orders of different amounts. How would the PHP script deal with a situation like this?

With your previous programming knowledge, you might recall that the PHP script, in this case, need not execute 23 times. The script will simply call a function that will calculate the discount for each book and return the cumulative result to the main program. Therefore, even if a customer buys 10 books the script doesn't look bulky and works as efficiently as ever.

In this chapter, you will learn about functions. You will learn how to declare and create your own custom functions and make them work for you.

Introduction to Functions

A *function* is a self-contained and independent block of code that accomplishes a specific task. Generally, this task might need to be repeated several times over. As a function is a self-contained chunk of code that can operate independently, it is not necessary that it be a part of a given script. This means that a function may be an integral part of a script or may be a part of an external library. If a function is a part of a given script, the code within a function is ignored until the function is called.

The major advantages of using functions while scripting in PHP (or in any other programming language, as a matter of fact) are:

- **Simplification of code**. Functions make the code very easy on the eye and therefore very easy to understand. Instead of writing the repetitive code as and where required in the script, you write the code once as a function and call it whenever you need to perform the task. As a result, functions also save your script from being needlessly bulky.
- **Reusability of code**. Once written, the function can be called not only from the script that contains the function but also from other scripts. This reduces the onus on you as a programmer to repeat the same functionality in different scripts.
- **Modularity of code**. To understand how functions also offer modularity, consider the example cited in the beginning of the chapter. Suppose that after some months the discount being offered on the books is 5 percent, regardless of the total purchase. In this case, you only need to change the code of the function instead of changing the code in many places.

So how does a function work? It's simple! Though you could code a function to behave differently, generally a function will:

1. Accept a value from the script that called it.
2. Process the result on the basis of code specified within it.
3. Return the result and the control to the caller script.

You should be aware of the main features of built-in PHP functions.

- The latest and the best feature of PHP 4 is that you don't need to define a function before you reference it. In simpler words, you can call (or reference) a function much earlier in the script and define it much later.
- If you have declared a function, you cannot redefine or undefine it. This is because PHP does not support function overloading. Therefore, you have to define each function with a unique name.

  Note Function overloading is a feature of functions in other programming languages, such as C and C++, that allows the creation of multiple functions with the same name.

- PHP 4 does not allow functions to support a variable number of arguments.
- PHP 4 supports default arguments.

You'll get familiar with all of these features as you progress with the chapter.

Next you'll look at the types of functions PHP has to offer you.

Types of Functions

PHP offers two types of functions. These are categorized as:

- Built-in functions
- User-defined functions

A close look at these functions follows.

> **Note** Some experts also refer to a third category of functions, namely variable functions. You'll learn about these functions later in this chapter.

Built-in Functions

Built-in functions are predefined functions that are available with PHP and behave in a predetermined manner for which they were created. There are hundreds of built-in functions that you can use. Some of the most commonly used built-in PHP functions that you are already familiar with from the previous chapters include `count()`, `echo()`, `array_diff()`, `array_reverse()`, etc.

In contrast to the built-in functions provided by PHP are the user-defined functions that are entirely dependent on the programmer's discretion.

User-Defined Functions

Also referred to as custom functions, *user-defined functions* are not provided by PHP. Rather, they are created by the programmer-that is, you. Since you create them, you have complete control of these functions. As a result, you can make a function behave exactly the way you want it to. For this reason, user-defined functions are quite popular in the programmer community.

How can you create your own functions in PHP?

Declaring a Function

The syntax that you would use in PHP is much the same as in other programming languages. Following is the syntax you would use to declare a function in PHP:

```
                  function function_name
($argument 1, $argument 2, $argument
...  ... .., $argument n)
                {
                        //code
                }
```

In the above syntax:

- **`Function`** is the keyword used to declare a user-defined function.
- **`Function_Name`** is the name of the function that you want to create. This would be the name by which you'll later reference (or call) the function. The function name should be unique because PHP doesn't support function overloading. When naming a function, you need to follow the same rules as in variable-naming conventions. However, the function name cannot start with the $ character, as variables do.
- **`Argument(s)`** is the value that you pass to the function. As you can see from the above syntax, a function can have multiple arguments, separated by commas. However, arguments are optional. You can

choose not to pass an argument while calling a function.

Caution Remember that an argument is always enclosed within parentheses (). Even if you declare a function that takes no arguments when it is called, you still would need to use the parentheses, as in `Function_Name()`.

- **Code** is the set of statements that are executed when the function is called by the main program. The output of the function is the result of this code. If the number of statements is two or more, the code must be included within curly braces ({}). However, you don't need to use these curly braces if the code part of the function contains only one statement.

On the basis of the syntax specified above, you will now learn to create simple argument-less functions.

Argument-less Functions

Following is the example of a simple 'argument-less' function.

```
<?php
  function no_arg() //Declaring the function
called no_arg
 {
        echo "Hello World! This is the output
of the function.", "\n"; //Function code
 }
  echo "The function has not been called yet.", "\n";
  no_arg(); //Calling the function from the main script
  echo "The function has been called!", "\n";
?>
```

In the above example, a function `no_arg` has been declared. Since no arguments have been specified in the parentheses following the function name, `no_arg` is an argument-less function. When the function is called with the help of the statement `no_arg();`, it displays the message `Hello World! This is the output of the function.` on the screen. The output of the function is given in Figure 6-1. Now look at how to declare a function with arguments. In this section, you'll also look at what it means to pass parameters to a function and implement the knowledge in your custom functions.

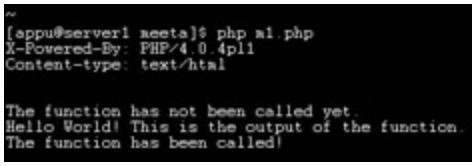

Figure 6-1: Output of an argument-less function

Passing Arguments to Functions

Arguments can be passed to functions in PHP in the following three ways:

- Passing default argument values
- Passing arguments by values
- Passing arguments by reference

You'll focus in detail on each of these argument-passing methods in the following sections.

Passing Default Argument Values

In this method, the function must take an argument when it is called. However, if no value is passed when the function is referenced from the main script, a default value is assigned to the function argument. Following is the example of a function that takes default arguments:

```
<?php
  function counter( $number = 6) //Declaring a
function called "counter"
 {
        for (; $number < 10; $number++)
        {
              echo $number, "\n";
        }
 }
  echo "The function has not been called
yet.", "\n";
  counter(8); //Calling the function and
passing the value "8"
to the argument of the function
  counter(); //Calling the function and
passing the default
value (6) to the argument of the function
  echo "The function has been called!", "\n";
?>
```

In the above code, the `counter` function takes the argument `number`, which has been assigned a value of `6`. When the function is called and is passed a value of `8` (`counter(8)`), the default value of the argument is overridden and is substituted by `8`. The function is then executed accordingly. On the other hand, when the function is called and passed no value (`counter()`), the function takes the default value of the argument and is executed accordingly. The output of the above code is shown in Figure 6-2:

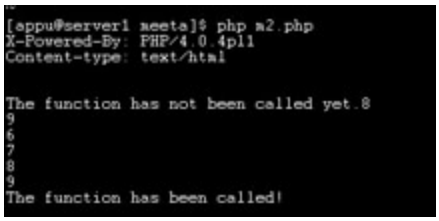

Figure 6-2: Passing default argument values

Now consider the second method of passing parameters to functions in PHP.

Passing Arguments by Values

This is the default method of passing values to functions in PHP. In this method, a value (or parameter) must be passed to the function when it is called from the main program. Following is the example of a function that takes arguments when it is called:

```
<?php
  function counter( $number) //Declaring
a function called "counter" that requires a value to be
passed to the argument called "number"
 {
        for (; $number < 10; $number++)
        {
              echo $number, "\n";
        }
 }
  echo "The function has not been called
yet.", "\n";
  counter(3); //Calling the function and
passing the value "3" to the argument of the function
  echo "The function has been called!",
"\n";
?>
```

In the above example, a function called `counter` has been declared, which takes an argument called `number`. When you call this function, you must pass a value to the argument of the function. Otherwise, PHP displays an error message. When the counter function is referenced from the main script (`counter(3)`), the value 3 is passed to its argument `number`. Each time the `for` loop runs, it increments the value of `number` by 1 and prints the result. The output of the above example is shown below in Figure 6-3.

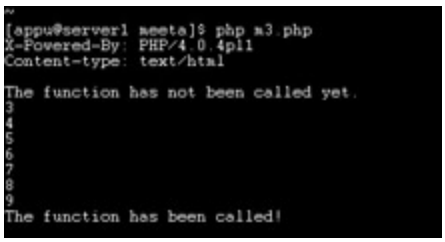

Figure 6-3: Passing argument by values

The next section discusses the third method of passing values to a function when it is referenced from the main script.

Passing Arguments by Reference

While passing arguments by value, only copies of arguments are passed to the called function. As a result, any modifications done to these values within the called function do not affect the original value in the calling function. Consider the example below:

```
                <?php
                  $num = 10;
                  function called_function( $number)
//Declaring a function that requires a value to be passed
to the argument called "number"
                 {
                         $number = $number+1;
                        echo $number, "\n";
                 }
                  called_function($num); //Calling the
function and passing the argument "num" to the function
                  echo $num, "\n";
                ?>
```

In this example, the value of the variable `num` is passed to the `called_function` and is stored in the local variable of `called_function`, which is `number`. The original value passed to the function is incremented by 1 and is displayed. When the control returns to the caller (the main script, here), the original value of `num` is displayed. This shows that although `num` was passed to another function, its value remained unchanged in the main script. The output of the above code is shown below in Figure 6-4.

Figure 6-4: Not passing argument by reference

Sometimes, such as in a script used for complex mathematical computations, you might want a parameter to be passed to a function and its new value, instead of the older value, to be returned to the calling program or the main script. As you saw in the above example, this is not possible if you pass the argument by its value. However, by passing arguments by reference, you can allow the called function to modify the value of an argument passed to it and return the modified value to the calling function. Arguments are passed by reference by using the sign & before the argument. Consider the following example. It is almost the same as the above example. However, you'll be passing the argument by reference instead of value.

```
                <?php
                  $num = 10;
                  function called_function( $number)
//Declaring a function that requires a value to be
passed to the argument called "number"
                 {
                         $number = $number+1;
                        echo $number, "\n";
```

```
                }
                 called_function(&$num);
//Calling the function and passing the argument
"&num" to the function
                   echo $num, "\n";
               ?>
```

In the preceding example, the argument `num` is passed by reference. Therefore, instead of a copy of the argument, a reference to the memory location of the argument is passed. As a result, `num` is modified by `called_function`. The output of the above code is shown below in Figure 6-5.

Figure 6-5: Passing argument by reference

Returning Values from Functions

Functions are also capable of returning a value to the calling function or the main script. In PHP, as in most other programming languages (such as C++), the `return` statement is used to return the value of a variable to the main script or the calling function. Consider the following example:

```
<?php
  function sum( $num1, $num2) //Declaring
a function called sum
 {
        $result = $num1 + $num2;
       return $result;
 }
  echo sum(233, 19), "\n";
?>
```

In the above code, the function named `sum` has been declared. This function, as the name suggests, is used to calculate the sum of two numbers supplied by the main script. The variable `result` within the function is used to store the sum of the two numbers that are passed to the function, when the function is called. With the help of the `return` statement, the result of the computation is returned to the main script, which in turn is displayed on the screen. The output of the above code is shown below in Figure 6-6.

Figure 6-6: Returning values to functions

UNIT - 3
UNDERSTANDING CLASSES, COOKIES, HANDLING FILES

Chapter 7: Understanding Classes

Overview

You hear a lot about object-oriented programming. What is it? And most importantly, why do you have to talk about it at all, if you are learning to write scripts in PHP? Well, here I will not go into the depths of object-oriented programming (which is very deep indeed!), but you need to have some idea about object-oriented programming. Although PHP is not a true object-oriented language, many aspects of object-oriented programming have been incorporated into it, more so in PHP 4.

The basic concept of object-oriented programming is that programs should be modeled on real life. In simple words, your PHP scripts must reflect the real world, which they are meant to serve. In the real world, you deal with objects. But look at the world around you. It is full of objects of different shapes, sizes, and colors; objects stationary and mobile; objects that exist in this world for different purposes. If you were to deal with this vast number and variety of objects simultaneously, you would be at a loss. So you can only do what the zoologists do-classify them on the basis of their attributes. Similarly, the millions of objects that you see in this world can be classified into various groups on the basis of common attributes that they possess. These groups are known as *classes*.

In this chapter, you will learn about the concept of classes and the role they have to play in making your PHP scripts survive in the real world. You'll learn how classes are created and how classes inherit functionality from other classes.

Classes

In object-oriented terminology, a *class* is defined as a set of objects that share the same characteristics and display common behavior. Along the same lines, an *object* is defined as an entity that belongs to a class, can be uniquely identified, and exhibits certain behavior that is common to the entire class.

To understand and relate the concept of classes and objects to the real world, consider a simple example from the animal kingdom. The peacock, the sparrow, and the kingfisher are all called birds. Why? Because all of these share some common characteristics. All of them lay eggs and hatch their young ones; all of them are covered with feathers, have hollow bone structures, and have the ability to fly. Thus, you may say that the peacock, the sparrow, and the kingfisher all represent unique objects that share structural and behavioral similarities and belong to the class called birds.

In PHP, a class is a collection of variables and functions. The class variables contain the data that is required for the functionality of the class. Class functions, on the other hand, operate on these class variables and implement the functionality of the given class. An object is an instance of a class that is used to initialize the class.

> **Note** In PHP literature, class variables are also commonly referred to as *member variables* or *properties*. Similarly, class functions are also known as *methods*.

In the following section, you will learn how to create a class and its objects in the PHP scripting language.

Creating a Class

In PHP, you can define (or create) a class using the following syntax:

```
                  class class_name //Declaring a class
                  {
                  var $variable_name; //Declaring a class variable
function function_name($argument 1, [...][...], $argument n)
//Declaring a class function
                 {
                        //function code
                 } //The class function ends here
                 } //The class ends here
```

You'll now create a simple class, using the syntax specified above.

```
                  <?php
                  class PrintName
                  {
                        var $name;

                        function show_name()
                        {
                             echo "\n";
                             echo "The name passed to
this method is $name.",
```

```
"\n";
                              echo "Hi $name! How are
you doing?", "\n", "\n";
                        }
                  }
                  ?>
```

In the above example, a class called `PrintName` has been created. This class consists of one member variable (or attribute) called `name`. The class also consists of a method `show_name`. This method, when called—as you'll see later—displays two messages with the value that the `name` variable receives during execution.

Note the keyword `var` before the member variable, `name`. Member variables of a class are always declared with the keyword `var`. This is because `var` declares the variables as a part (member) of a given class and hence allows member variables to be accessed from outside the class. If you forget to declare member variables of a class without the `var` keyword, you will end up with a parse error.

Now that you've learned to create a class in PHP, how do you work with it? Bear in mind that, as in other object-oriented programming languages, the data residing in a class (in the form of class variables) and class functions cannot be accessed from outside the class! If you remember your basic concepts in object-oriented programming, this feature is known to programmers as *encapsulation*. Encapsulation entitles the prevention of access to nonessential details within a class.

The member variables and methods are well hidden within the class and are inaccessible. So how will you access these attributes and methods from outside the class? You'll need to instantiate the class to do so. The next section throws light on the instantiation of a class.

> **Note** Note that nonessential does not mean unimportant! Nonessential, here, refers to information that is not required in a given case, but might be highly important in another circumstance.

Instantiating a Class—Making Use of Objects

A class is simply a blueprint. Therefore, it does not exist until you create at least one instance of it. In the terminology of object-oriented languages, this instance of the class is referred to as an *object*. After you've created an object of a class, you can access the member variables as well as the methods hidden within the class to which the object belongs. Consider the following example to understand how you can use the `name` variable and the `show_name()` method of the `PrintName` class that was designed in the preceding example.

```
                  <?php
                  class PrintName
                  {
                        var $name;

                        function show_name()
                        {
                              echo "\n";
                              echo "The name passed to this
method is $name.",
"\n";
                              echo "Hi $name! How are you
doing?", "\n", "\n";
                        }
                  }
                  $obj1 = new PrintName;   //Instantiating
```

```
the class called "PrintName" with the object "obj1".
                $obj1 -> name = "George";   //Setting
the value of the class variable "name" to "George".
                $obj1 -> show_name();   //Calling the
method "show_name()".
                ?>
```

In the above example, an object—`obj1`—of the `PrintName` class is created. This object instantiates the `PrintName` class. This means that the class is now allocated memory and you can now access the variables and methods hidden within it with the help of the `obj1` object. `Obj1` uses the `->` operator to access the `name` variable and `show_name()` method within the class. `Obj1` sets the value of the `name` variable to `George` with the help of the statement `$obj1 -> name = "George"`, and then calls the `show_name()` method, which displays the messages `The name passed to this method is George.` and `Hi George! How are you doing?`

Note If you would have tried to set the value of the `name` variable to `George` and call the `show_name()` method without the object `obj1` of the `PrintName` class, PHP would have displayed an error.

Figure 7-1 shows the output of the preceding example.

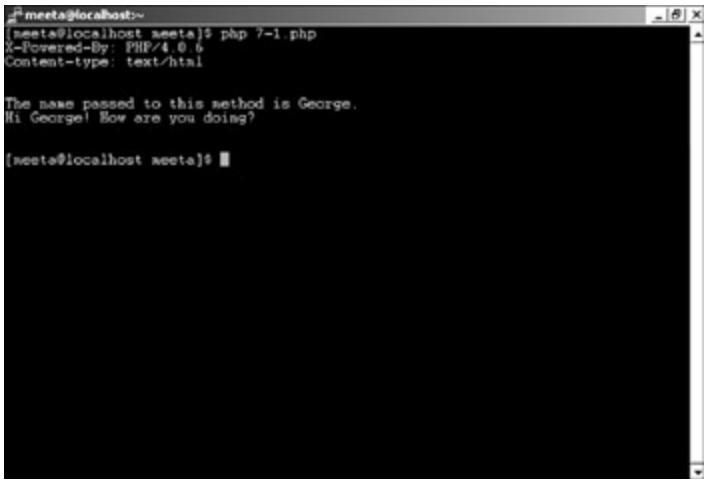

Figure 7-1: Output of the instantiation of a class

You have learned how to access member methods of a class by using objects. Let me tell you how you can access these hidden methods without using an object at all!

Accessing Class Methods without an Object

You can access methods hidden within a class without using an object, with the help of `:: notation`. Consider the following example to understand how you can do so.

```
<?php
class area
{
      function calculate_area($length, $breadth)
      {
           return $length * $breadth;
      }
```

```
}
$result = area::calculate_area(225, 56);
 echo "The area Is: $result.";
?>
```

In the above code, the use of `:: notation` provides direct access to the `calculate_area()` method of the `area` class. The result is calculated on the basis of values supplied at the time the method is called and is returned to the `result` variable. The output of the above code is `The area is 12600.`

You've just seen how to access class variables and methods from outside the class in two ways. But how would you access a class variable or method from within the class? Unlike C and C++, PHP doesn't allow you to access class variables and methods by simply using their names even in the class where they were created (or declared). The next section shows you how to get around this problem.

The *$this* Variable

PHP uses a special variable called `$this` to access a class variable (declared with the help of the `var` keyword) from within a given class. The use of the `$this` variable in PHP is the same as creating an instance of the class within the class. It is unlike an object that creates an instance of a class outside a class. The preceding example has been changed a little to demonstrate the use of `$this`.

```
<?php
class PrintName
{
      var $name = "George";

      function show_name()
      {
            echo "Hi $this->name! How are
you doing?", "\n",
"\n";
      }
}
$obj1 = new PrintName;
$obj1 -> show_name();
?>
```

In the above example, the name variable has been referenced directly within the `PrintName` class by using the `$this` variable. The output of the above code will be `Hi George! How are you doing?`

Note In some literature, if you come across the `$this` variable as being referred to as an "object," do not be alarmed. The terms "special variable" and "object" mean the same in reference to `$this`.

You can also change the value of a member variable within a member method, as is demonstrated in the following example.

```
<?php
class PrintName
{
      var $name = "George";

      function show_name()
      {
            echo "\n", "\n";
```

```
                              echo "Hi $this->name! How are
you doing?", "\n",
"\n";
                        }
                        function change_name($somename)
                        {
                              $this->name = $somename;
//Changing the value of
the "name" variable from "George" to

//the new name supplied to the function.
                        }
                  }
                  $obj1 = new PrintName;
                  $obj1 -> show_name();
                  $obj1 -> change_name("Amanda");
                  $obj1 -> show_name();
                  ?>
```

In the above example, first `Hi George! How are you doing?` will be displayed because the `show_name()` method is being called by `obj1` and `George` is the default value of the `name` variable. The statement `$obj1 -> change_name("Amanda");` causes the `change_name()` method to be called, which resets the value of `name` to `Amanda`. The `show_name()` method is then called, which in turn displays `Hi Amanda! How are you doing?`, as is shown in Figure 7-2.

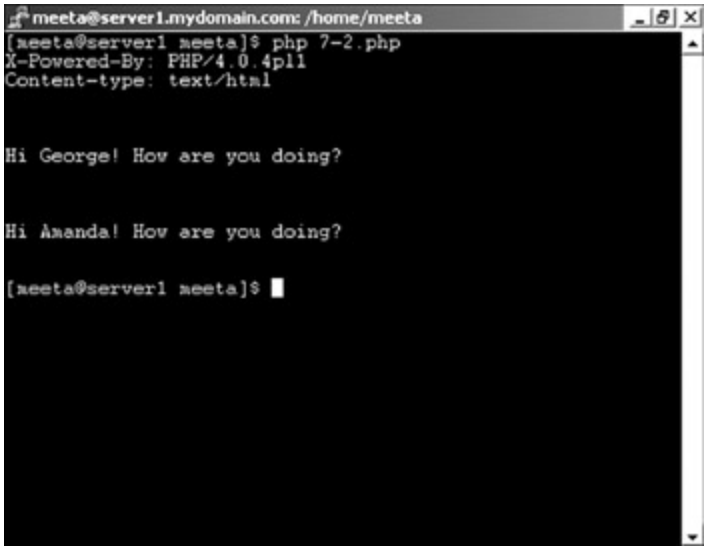

Figure 7-2: Changing the value of a member variable within a member method

As you might have realized, a member method of a class, like normal functions, is not called until an object accesses it explicitly. It means that until you write the `$obj1 -> show_name()` statement in your code, the `show_name()` method will not be executed. In the next section, you'll learn about class methods that are executed automatically the moment you instantiate a class (that is, create an object of the class).

Constructors

A *constructor* is a special member method of a class, which is executed automatically when you create an object of the corresponding class. The constructor, by rule, must always have the same name as the class. Therefore, you can have only one constructor within a class. The following example demonstrates the working of a constructor.

```
<?php
class PrintName
{
      var $name;

      function PrintName()
      {
            $this->name = "Angelica";
            echo "\n", "\n";
            echo "Hi $this->name! How are
you doing?", "\n",
"\n";
      }
      function show_name()
      {
            echo "Hi $this->name! How are
you doing?", "\n",
"\n";
      }
}
$obj1 = new PrintName;
$obj2 = new PrintName;
?>
```

As shown in Figure 7-3, the output of the above code would be `Hi Angelica! How are you doing?` twice. This is because the `PrintName` method is executed each time you create an object of the `PrintName` class.

Caution Remember that the name of a constructor is case-sensitive. In the preceding example, a method called `printname` or even `Printname` would not be treated as a constructor by PHP.

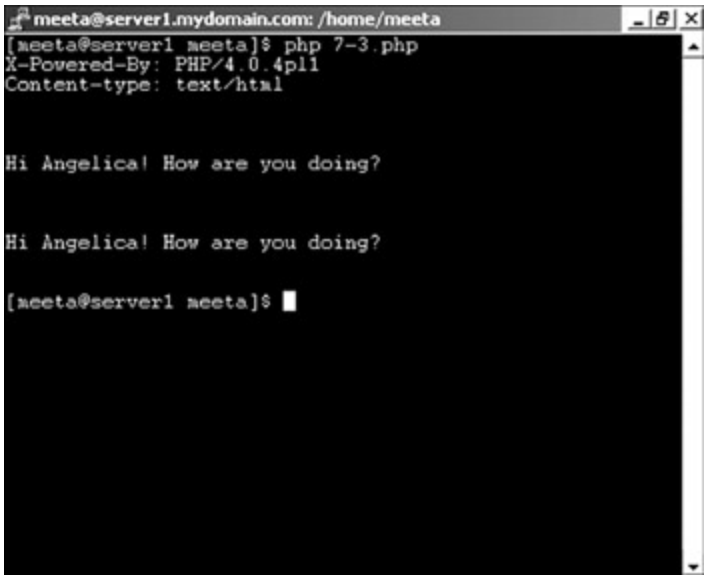

Figure 7-3: Working of a constructor

In your previous forays with other object-oriented programming languages, you must have come across the concept of *destructors*. A destructor is a special method that is automatically called to free memory space when an object of a class is destroyed. For example, an object might be destroyed if it goes out of scope. However, PHP does not support explicit destructors. You can use the `register_shutdown_function()` to call your own customized function, where the code to free objects is written.

In the next section, you'll learn about an important aspect of classes-inheritance.

Chapter 24: Cookies

Have you tried accessing Yahoo Mail or any other Web-based mail? Most of these sites prompt you to remember your login ID, so that next time when you log in to the site you are only prompted for the password. Have you ever wondered how a site remembers your login ID and other details? The answer is cookies (not the ones you eat, by the way!).

In this chapter, you will learn about cookies and the role they play in maintaining the state between your subsequent visits to a Web page. You will learn to implement cookies in your Web site by setting and creating them. You'll also learn to access and delete cookies. And you'll learn why there is such furor around the use of cookies among the Web community and how you can prevent loss of sensitive data.

What Is a Cookie?

You already know that the Internet is based on Hypertext Transfer Protocol (HTTP), which is a stateless protocol. This implies that once a transaction between the client machine and the Web server is finished, the Web server loses all the memory regarding the transaction.

Let me try to explain this concept in simple English. Suppose a user visits a commercial site, adds three books to the shopping cart, and before making the payment for the books, moves on to another site. After a while the user returns to the same site. Will the user need to add the three books chosen earlier again to the shopping cart? The answer is yes because as I said earlier, HTTP is stateless and therefore loses any memory of the earlier actions. This means the user will have to search for the same books again and add them to the shopping cart and pay for the books before moving on to the next site. However, the answer to the same question can be no, the user would not have to add the books again, if the site uses cookies. So what is a cookie?

A cookie is a small piece of information (or a message, if you like) that a Web server can store through your Web browser on to your hard disk when you visit the corresponding site. The Web server can also retrieve this information later when you visit the same site next time.

When you visit a cookie-enabled Web site, you might need to log in to the site or register using a password and other relevant information. This information is stored into a small text file whose maximum size is 4 KB. This file is referred to as a cookie and contains the relevant user-related information, such as User ID, password, list of pages that the user visited, and the date the user last visited a page.

For example, say you searched for some book on the well-known online book store Amazon, at **http://www.amazon.com/**. This is a cookie-enabled site, and Figure 24-1 depicts the cookie that was created on your local hard disk.

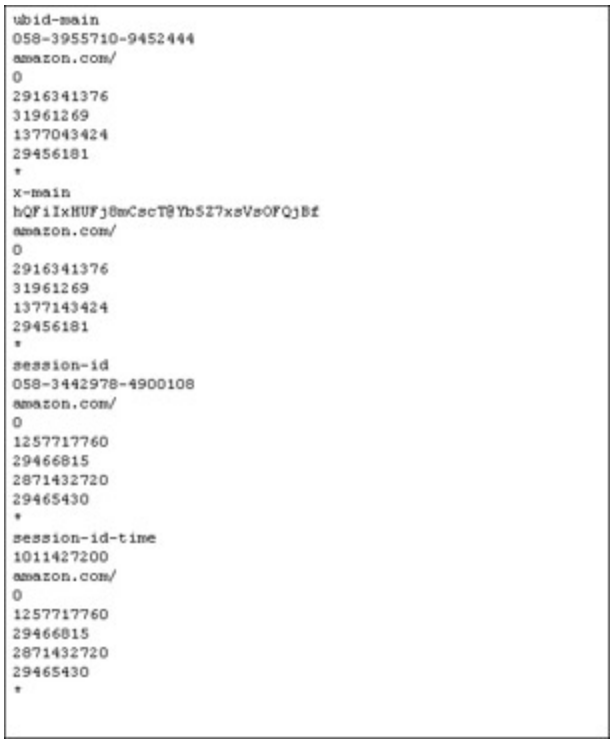

Figure 24-1: A sample cookie

In Figure 24-1, `058-3955710-9452444` is the unique ID that will be assigned to you when you visit the Amazon site for the first time. The site also stores a session ID, in this case `058-3442978-4900108`, that uniquely identifies the given session along with the time when the given session started. As shown in Figure 24-1, the session ID time is `1011427200`. A main ID, `hQFiIxHUFj8mCscT@Yb5Z7xsVsOFQjBf`, is also stored for the internal processing by the site. The rest of the information in the cookie is considered the housekeeping information for your browser.

As you saw in Figure 24-1, the information that is stored in a cookie is in the format of name and value. Consider the following name-value pair:

`ubid-main 058-3955710-9452444`

In the above example, `ubid-main` is the name and `058-3955710-9452444` is the corresponding value.

Now that you have the basic idea what a cookie is, let me explain what role it has to play and how it helps while you browse the Internet. Continuing the same example I cited above, suppose you visit an online bookstore and add three books to your shopping cart. The list of all the items that you picked up is stored in the cookie that was created automatically. You might continue browsing the site after adding books to your shopping cart. When you are through browsing the site, you would now like to pay for all the items that you picked. When you return to the shopping cart, all the items that you added to it are very much there. Make the payment and the transaction is over! Simple, isn't it? Cookies can make your life this easy.

Most of the sites that are created nowadays feature cookies. The use of cookies is popular among the Web community for the following reasons:

- To determine how many users visit the given Web site and how often. These statistics are especially helpful if the site uses proxy servers that make keeping a tab on visitors a problem.
- For storing details of the users who visit the site or register on the Web site. However, since the cookie

stores the details at the client's hard disk, extra details need not be stored in a database.

- For helping a site store individual user preferences at the client end, thus allowing users to customize the interface (such as layout and colors) as per their liking.
- To prevent repetitive logins, thus making the login process faster. In addition, since the cookie is stored at the client end, the Web server need not be burdened each time a user needs to log in to the site. The server only needs to authenticate the first-time users.
- For tracking a user's path and activities on a given Web site. In addition to determining the general preferences of users, this feature allows the Web administrators to track miscreants.
- For generating individual user profiles. For example, some sites display personalized messages to their users when they log in to the site.
- For storing the items selected by the site users in their respective shopping carts.

Now that you have the basic idea of what a cookie is and the reason why it is used, let me discuss how a cookie works.

How Does a Cookie Work?

A cookie works in the following manner:

1. When you type the URL of the destination Web site in the Address bar of your browser, the address is located and if found successfully, a request is sent to the Web server that hosts the site.
2. If the Web server accepts the request, the Web browser at the client end checks for an existing cookie from the given site.
3. If the cookie is found, the browser sends all the name-value pairs in the cookie to the server as the HTTP header. In addition, the expiration date of the cookie, if any, and a path is also sent to the server along with the name-value pairs. The expiration date indicates the date and time when the cookie will be considered invalid. The path helps the Web server to associate various values in the cookie to different pages of the site that is being accessed. This information (name-value pairs, expiration date, and path), when received by the server, is then used by the server for internal use and other activities, such as validating the user.
4. If the corresponding cookie is not found on the local hard disk, the server is notified about the absence of a cookie. In this case, the server generates a new ID for the client who requested a connection and sends the cookie containing the name-value pair(s) to the requester's Web browser. The browser then stores this cookie on the hard disk of your machine.

The communication between the client and server using cookies is depicted in Figure 24-2.

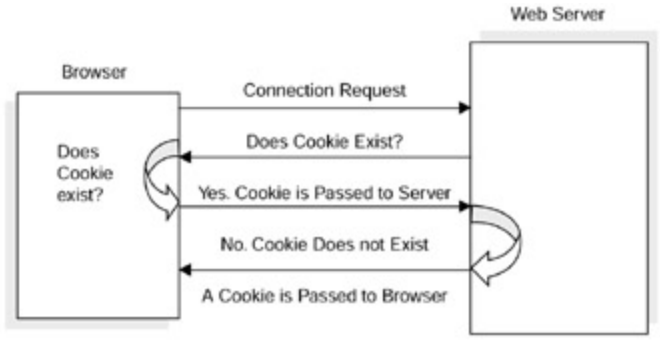

Figure 24-2: Communication between server and browser using cookies

In the next section, you will learn about the parameters that determine the validity and the longevity of a cookie.

The Scope of a Cookie

As you already know, a cookie is a name-value pair. In addition to the name and value parameters, several other parameters are used, which allow you more control over your cookies. For example, you can control the time span for which a cookie will be considered valid. Similarly, you can also control the type of information it will provide and the security parameters it will support. In other words, these parameters define the scope of a cookie.

The parameters that are used to determine the scope of a cookie are:

- **Expiration parameter:** This parameter defines the validity, life span, and longevity of a cookie. It determines the date and time of expiration of a cookie, where the time is specified in the GMT format. After a cookie reaches its expiration date and time, it can no longer be sent to a client.

  Note The default value used for this parameter is `Until the browser is closed`. This implies that the cookie survives only until the Web page in the browser window is open. The moment you close the browser window on the client machine, the cookie is discarded and becomes unavailable next time you run the browser.

- **Path parameter:** This parameter is used to limit the scope of a cookie to a certain part of the document tree within the Web server. In simpler terms, this parameter specifies the path to all files and directories on a Web server for which the cookie is considered valid. By setting the path scope within a document hierarchy, the cookie is sent to the Web server only if the client requests a page that exists in the specified path.

  Note The default value of this parameter is `/`, which implies that the given cookie will be available by default to any pages in the same directory as the page that created it or in pages lower down in the hierarchy.

- **Domain parameter:** This parameter is used to specify the domain for which the cookie is valid. The cookie can be limited to a particular host by specifying the name of the host as `singlehost.com`. Similarly, a cookie can also be made valid for an entire domain by specifying the value of this (domain) parameter as `.domainname.com`. The `.` in the domain value implies that the cookie is not limited to a specific host within the domain.

  Note The default value of the domain parameter is the domain name of the server that generated and set the cookie on the client's hard disk. This value is generated automatically.

- **Security parameter:** This parameter ensures that the confidential data stored in the cookie is safe from unauthorized access while it travels from the Web server to the client machine. By enabling this parameter a secure channel, such as Secure Hypertext Transfer Protocol (HTTPS), is used for exchanging cookies between the server and the client. If the value of this parameter is not specified, the cookie is sent through an unsecured channel.

You must remember that the cookie name and value parameters are mandatory for the definition of a cookie. The rest of the parameters-expiration, domain, security, and path-are optional.

Figure 24-3 depicts a cookie that contains all the parameters, mandatory as well as optional, that a cookie can take.

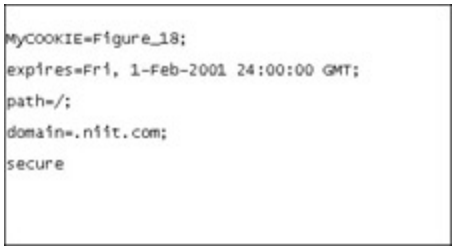

Figure 24-3: A cookie with all parameters

The next section throws light on the restrictions that you might face while using cookies.

Note The default value of the secure parameter is `disabled`. As a result, all the data and cookies are exchanged between the two communicating ends regardless of the fact whether the transmission channel is secure or not.

Restrictions While Using Cookies

Since a cookie might contain confidential information, such as login ID and password, there are various restrictions on the use of cookies to prevent their abuse. These restrictions include:

- The maximum allowed size of a cookie is 4 KB. This is because cookies are stored on the user's hard disk. If the size of a cookie is large, it is possible that cookies from various Web sites a user visits in a day might fill up the client's hard disk.
- A Web server cannot store more than 20 cookies on a user's computer. This again ensures that one Web server does not fill up the client's hard disk.
- A browser at the client end can store a maximum of 300 cookies.
- Only the Web server that issued a cookie to a client can read the data stored in the cookie. Although all cookies are stored in the same folder, a Web server cannot access or read cookies that have been created by other Web servers or domains.

Now that you know how a cookie does its work and you know about its scope, you will learn about the implementation of cookies in PHP.

Implementing Cookies in PHP

Implementation of cookies in PHP is very simple. This is because cookie support is fully integrated in PHP. For example, cookies are automatically considered as global variables and can be accessed from anywhere in the PHP script. Also, they are accessed and read as normal variables. In this section you'll learn to deal with various aspects of cookies in PHP. These include:

- Creating a cookie
- Accessing a set cookie
- Using multiple-value cookies
- Deleting a cookie

Creating Cookies

You need three basic functions to create a cookie. These include:

- `setcookie()`
- `time()`
- `mktime()`

Let us now learn how to create a simple cookie using the above functions.

The *setcookie()* Function

You can create a cookie in PHP by using the `setcookie()` function. The Web server uses this function to set a cookie and send it to the client when the client requests a connection.

Caution If you have used the `setcookie()` function to set a cookie, you will need to call this function before any `<html>` or `<head>` tag. Otherwise, you will encounter an error message. This is because cookies must be sent before any HTTP headers during a connection. The logic behind this fact is that a cookie is set as part of the HTTP header. As a result, it can't be set after the header has already been sent!

Another important thing that you need to remember while using the `setcookie()` function is that even a single whitespace or a newline character before the PHP tag in your script can cause your script to malfunction.

The syntax of the `setcookie()` function is specified below:

```
setcookie (string name, string value, int expiry, string path, string
domain, int secure);
```

As the above syntax specifies, the `setcookie()` function takes six arguments. These include:

- **name:** This argument specifies the name of the variable that will hold the corresponding value in the name-value pair. This is a global variable. As a result, this variable is accessible in the subsequent PHP scripts.

- **value:** This argument holds the value of the variable specified by the `name` argument.

  Note If you set a cookie with an invalid name (a name value that contains invalid characters), the Web server automatically changes the name of the cookie. For example, if you happen to set the name of a cookie as My%Cookie, the name will be automatically converted to My_Cookie.

- **expiry:** This argument specifies the time after which the cookie will be considered invalid and thus rendered unusable. The value of this argument is specified in Greenwich Mean Time (GMT). As mentioned earlier, if you do not specify the value of this argument, the cookie is considered to be valid as long as the browser window at the client side remains open. With the closing of the browser window, the cookie expires.

  Note The value of the `expiry` argument is specified by two functions, `time()` and `mktime()`, about which you'll learn a bit later in this chapter.

- **path:** This argument, as you learned earlier, specifies the hierarchy of files and directories on the Web server for which the cookie is considered to be valid. The default value of this argument is `/`. As a result, if you do not specify the value of the `path` argument in the `setcookie()` function, the cookie will be considered valid for the entire files and directories on the Web server. However, if you specify the path, the cookie will be valid only for the files and subdirectories located in the specified directory.

- **domain:** This argument is used to specify the host or domain name for which the cookie is considered to be valid. If no value is specified for this argument, the host (Web server) that issued the cookie is considered to be the default value of this argument.

- **secure:** This argument determines whether the cookie must be transferred through a secure channel. If no value is specified for the argument, an insecure channel is used to transfer the cookie. However, if the value of this argument is `1`, the cookie is transmitted using the HTTPS protocol that ensures secure transactions over the Internet.

All arguments except the `name` argument in the `setcookie()` function are considered to be optional. Except for `expiry` and `secure`, which are integer values, all the other arguments can be bypassed by specifying `""` (empty string). In order to skip the `expiry` and `secure` arguments, you'll need to use `0`.

You'll need two more functions, `time()` and `mktime()`, to create a cookie. You'll learn about these two in the following section.

The *time()* and *mktime()* Functions

You can use the `time()` and `mktime()` functions to set the life span of a cookie to avoid the expiry of the cookie when the browser window closes.

The `time()` function is used to determine the current time. It returns time in seconds. The syntax of the `time()` function is specified below.

```
int time();
```

The `mktime()` function, on the other hand, accepts units of time, such as hours, minutes, seconds, months, and years, as parameters and is used to convert this information into a time stamp. Just like the `time()` function, the time stamp returned by the `mktime()` function is in seconds. The syntax of the `mktime()` function is specified below.

```
int mktime(int hour, int minute, int second, int month, int year,
int isDaylightSavingsTime);
```

In the above syntax, as the respective names of the variables suggest, `hour` accepts an integer value and is

used to specify the number of hours. Similarly, `minute` and `second` specify the number of minutes and seconds, respectively. The `month` argument is used to return the specified month, and the `year` argument is used to denote the specified year.

The last argument of the `mktime()` function is interesting. It is used to set the daylight saving according to the time zone you specify while installing PHP on your computer. The default value of this argument is `-1`, which implies that the value of this argument is unknown. If this argument is set to `1`, the returned time includes the daylight saving settings, and if the value is set to `0`—no points for guessing!—the daylight saving option is off.

> **Note** You can omit all the arguments of the `mktime()` function. In such a case, the values from the current date and time will be picked.

Now that you have basic knowledge of all the functions that are used to set a cookie, consider the following code.

```
<?php
$name = "MyCookie";
$value = "This is my first Cookie!";
$expiry = mktime(0, 0, 0, 7, 1, 2002);
$domain = "localhost.localdomain";
$setcookie($name, $value, $expiry, /, $domain, 0);
?>
<html>
 <head>
 <title> Testing the Cookie </title>
 </head>
  <body>
  <br> <br> <br>
  <h1 align = center> This appears when the cookie is created
successfully! </h1>
  </body>
  </html>
```

In the above code snippet, a cookie called `MyCookie` is created. Since the value of the `value` variable is `This is my first Cookie!`, this will be the text that is stored in the cookie. The expiry information defined by the `expiry` variable is `mktime(0, 0, 0, 7, 1, 2002)`, which implies that the given cookie will expire at midnight on July 1, 2002. The cookie can access the entire hierarchy of files and folders on the Web server (denoted by the `domain` variable) because the path variable is set to `/` in the `setcookie()` function.

> **Note** Note that in the above code, I have used the default values of the `path` and `security` variables. Therefore, the `setcookie()` function in the above code can simply be rewritten as `setcookie(($name, $value, $expiry, $domain)`. It will work just fine!

Figure 24-4 shows the result of the previous code snippet.

Figure 24-4: Creating a simple cookie

If you would like to see the use of the `time()` function, set the value of the `expiry` variable as `$expiry = time() + 86400;` in the above code. This will cause the cookie to be rendered invalid in 24 hours after it was first created. The `time()` function is used to determine the time (in seconds) the cookie was first created. `86400` (60x60x24 = 86400) is used to specify the number of seconds after which the cookie will expire.

In the next section, I discuss how to access a cookie.

Caution Before you execute the above code, you must ensure that your browser is set to accept cookies. Otherwise, you will end up with an error message. You can configure your browser to accept cookies in the security settings of your browser. For example, to enable Microsoft Internet Explorer to accept cookies, you need to choose the Internet Options option in the Tools menu of the browser window. Then click on the Security tab in the Internet Options dialog box and click the Custom Level button. Here you need to set the two cookie options—Allow cookies that are stored on your computer" and "Allow per-session cookies (not stored)"—to enable cookies.

Accessing Cookies

While working with cookies in PHP, after you have successfully set a cookie, the most important fact that you need to remember is that you cannot set and access a cookie in the same request. After you set a cookie, you'll need to reload the Web page before you can use it.

Consider the code given below.

```
<?php

if (!(isset($kookie)))
{
 $pagecount=0;
 setcookie("kookie",$pagecount);
 echo "<center> This is the first time you have accessed this
page in this session.</center>";
 echo "<center> A cookie was sent to you and stored in your
computer.</center>";
}
else
```

```
                    {
                     $pagecount=++$kookie;
                     setcookie("kookie",$pagecount, time() - 10 );
                     setcookie("kookie",$pagecount);
                     echo "<center>View Count :<b> " .$kookie.
"</b></center>\n<br>";
                    }

                    ?>
                    <html>
                    <head>
                    <title>Page title</title>
                    </head>
                    <body>

                    <center><b> Refresh button will refresh the page and the
page count! :-) </b></center>

                    </body>
                    </html>
```

In the above code, the function `isset()` is used to determine whether a cookie called `kookie` has been set or not. If the cookie has not been set, a variable `pagecount` is set to `0`. The same variable is also used to set the value of the cookie in the name-value pair to 0. Also, the messages `This is the first time you have accessed this page in this session.` and `A cookie was sent to you and stored in your computer.` are displayed. On the other hand, if this is not the first time the page has been accessed, the variable `pagecount` is incremented with the help of the statement `$pagecount=++$kookie;`. The cookie is then reset and a message depicting the number of times the page has been viewed is displayed on the screen. Every time you refresh the page (by clicking the Refresh button in the browser window), the view count is incremented by 1.

> **Note** The function `isset()` takes a variable as an argument and determines if the variable has been set or not. If the variable passed to the `isset()` function as the parameter is set, it returns True. Otherwise, it returns False.

Figure 24-5 shows the result when you execute the above code. Since the cookie has just been set, you'll not be able to access the cookie.

Figure 24-5: Viewing the Web page for the first time

Figure 24-6 shows the result of the preceding code after you have refreshed the Web page once. Now that the cookie was set when you first saw the page, this time you will be able to access the cookie.

Figure 24-6: Viewing the Web page after refreshing it once

Figure 24-7 shows the result of the preceding code after you have refreshed the Web page a few times.

Figure 24-7: Accessing the cookie after refreshing the page content a few times

You can use built-in PHP functions such as `header()` to reduce the burden of programming. Consider the following code snippet. This snippet is used to set a cookie and reload the same page that was used to set the cookie.

```
<?php

if (!(isset($kookie_new)))
setcookie("kookie_new", "A Short Cut");
header("Location: $PHP_SELF?kookie_new = 1");
exit;

?>
```

In the above code, the `header()` function is used to send a raw HTTP header string as a part of the HTTP header. The `Location` string that has been used above redirects the browser to an HTML page. By setting the corresponding URL for the `Location` string as PHP_SELF, the current page open in the browser window will be reopened automatically.

You can also use the `header()` function and `Location` string to redirect the browser to a different HTML page. Consider the code below to understand this.

```
<?php

if (!(isset($kookie_new)))
setcookie("kookie_new", "A Short Cut");
header("Location: test.html");
exit;

?>
```

Figure 24-8 shows the results of the above code snippet that opens `test.html`.

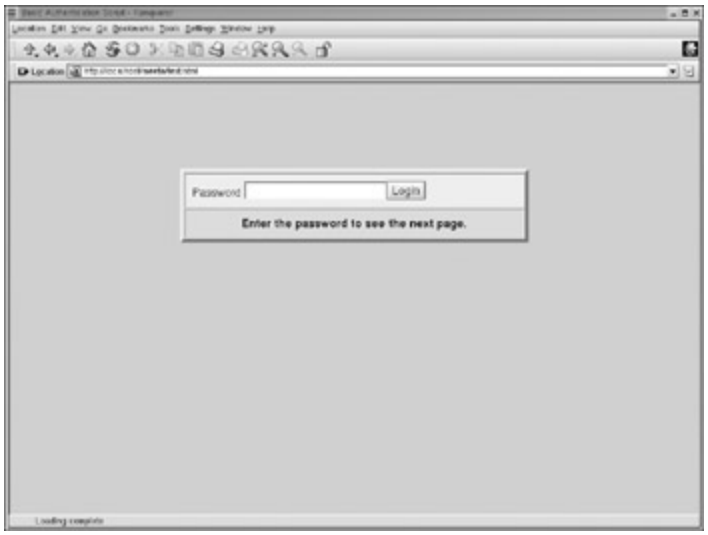

Figure 24-8: Redirecting the Web browser to another page using a cookie

After you have successfully set a cookie, you can access it by using one of the two methods specified below:

- **Using the name of the cookie as a variable.** When you try to access a cookie in PHP, a variable with the same name as the cookie is created. This variable name, then, can be used by your scripts. As a result, you can either reload the page that was used to create the cookie or load another page that uses the cookie. Suppose you created a cookie called `My_Cookie`. You can use the name of the cookie as a variable in your script as specified below:

  ```
  echo "My_Cookie";
  ```

- **Using $HTTP_COOKIE_VARS ["cookie_name"].** Again, the variable that is created by default in PHP by using the cookie name is also stored in the global array as `$HTTP_COOKIE_VARS` ["cookie_name"]. An additional advantage of using this global array is that it helps in distinguishing between different cookie names on the basis of their data sources, even if they happen to have the same name. You can use this global array in your PHP scripts as given below:

  ```
  echo "$HTTP_COOKIE_VARS ["My_Cookie"]"
  ```

Until now you have learned about single-value cookies. This implies that the cookies you have created until now have just held one value, such as page count or the Web page that your browser will be redirected to. However, sometimes a cookie may need to store multiple values. A shopping cart is a classic example of a cookie that stores multiple values. In the next section, you'll learn more about multiple-value cookies.

Multiple-Value Cookies

Now you must be wondering, "How is a shopping cart a multiple-value cookie?" The answer is simple! The cookie used for a shopping cart needs to keep track of various items that have been added to the shopping cart or the length of a session besides maybe the user ID, personalized settings, or the various pages that the user has visited in the current session. A whole lot of information to be stored in a small cookie, whew!

As the name suggests, the cookies that can store more than one value are referred to as multiple-value cookies. These cookies may prove to be very useful to you as a Web developer. As you might recall, the number of cookies that can be stored by a Web server on a single computer is restricted to 20! Therefore, instead of creating a bevy of cookies when a customer goes on a shopping spree and unnecessarily populating the user's hard disk, you can make wonderful use of multiple-value cookies.

To make use of multiple-value cookies, you need to specify a cookie as an array. As a result, every bit of information that you'll need to store as a part of one sin

gle cookie will be stored as an element of this array. Consider the code specified below that deals with multiple-value cookies.

```
                    <?php

                      $gmtseconds = time();
                      if (isset($kookie))
                       {
                        $pagecount=++$kookie[0];
                        setcookie("kookie[0]", $pagecount,$gmtseconds + 60);
                        setcookie("kookie",$kookie,$gmtseconds + 60);
                        setcookie("kookie",$gmtseconds,$gmtseconds + 60);


                        echo "Hi " . $kookie .  " !!<br>\n";
                        if ($kookie==1)
                         {
                          echo "You have seen this page for the first time
!<br>\n";
                         }
                         else
                         {
                          echo "You have seen this page " .$kookie[0]  .
" times.<br>\n";
                          echo "I remember U last saw this page ".
($gmtseconds - $kookie)." seconds ago";
}
                        exit;
                       }
                      else
                       {
                        if (isset($name))
                         {
                          $pagecount=0;
                          setcookie("kookie[0]", $pagecount ,$gmtseconds + 60);
                          setcookie("kookie", $name,$gmtseconds + 60);
                          setcookie("kookie", $gmtseconds,$gmtseconds + 60);
                          echo " The cookie is set ! Please press the Refresh
button to see what happens next.";
                           exit;
                         }
                      }

                    ?>
```

In the preceding code, the cookie called `kookie` is a multiple-value cookie. The first element of this cookie, `kookie[0]`, contains the latest page count (the number of times you have visited the page) with the help of the variable `pagecount`. The next element of this cookie, `kookie`, contains the name of the user. Similarly,

the third element, `kookie`, contains the time you last visited the site.

When you visit the page, the script checks if the cookie is set. Since you are visiting the page for the first time, the cookie will not be set. As a result, the script will set the cookie (`kookie[0]`) and record your name in `kookie` and the time (in seconds) when you visit the Web page in `kookie`. Also, a message will be displayed stating `The cookie is set! Please press the Refresh button to see what happens next.`

After you refresh the Web page, the script is executed again. The count is incremented with the help of the statement `$pagecount = ++kookie[0]` and new values—latest count and access time—are recorded in the array. Since now the value of the variable `pagecount` is `1`, the message `You have seen this page for the first time.` is displayed.

In case you have visited the page a couple of times (that is, refreshed the page a few times), the message `You have seen this page <number> of times.` is displayed, where `<number>` represents the value stored in `kookie[0]`. Another message, `I remember U last saw this page <time> seconds ago.`, is displayed. Here, <time> represents the value stored in `kookie`.

Figure 24-9 shows the output of the cookie when you log in to the page for the very first time.

Figure 24-9: Output of the cookie when you log in to the page for the very first time

Figure 24-10 shows the output of the execution of the cookie when you refresh the page for the first time.

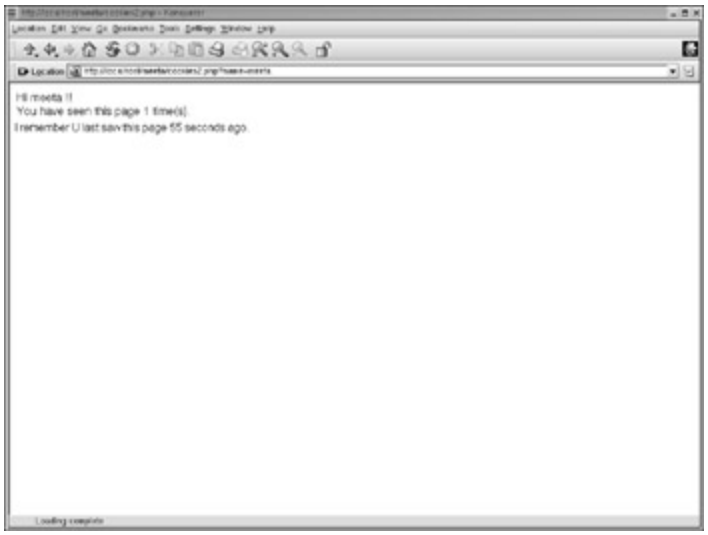

Figure 24-10: Output of the cookie when you refresh the page for the first time

Figure 24-11 depicts the output of the preceding code when you refresh the Web page three times.

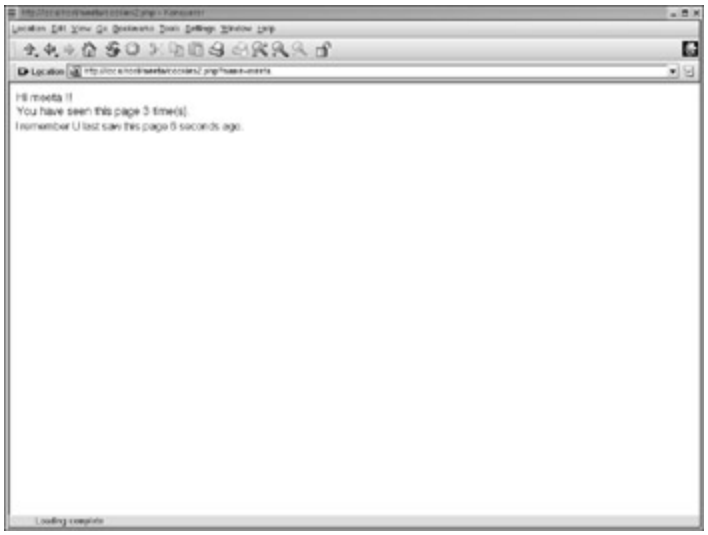

Figure 24-11: Output of the cookie when you refresh the Web page three times

The corresponding HTML part of the code that calls the PHP script (`cookies2.php`) specified earlier is as follows.

```
<html>
<head>
<title>Testing a Multiple Value Cookie</title>
</head>
<body bgcolor="blue" text="white">
<p>   </p>
<p>   </p>
<p>   </p>
<p>   </p>
<p>   </p>
```

```
<form name="nameform" method="GET" action="cookies2.php">
<center>
  <table>
    <tr>
      <td>
<b> Name </b>
</td>
      <td>

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

      </td>
      <td>
        <input type="submit" name="Submit" value="Login">
      </td>
    </tr>
  </table>
</center>
</form>

</body>
</html>
```

The above HTML code generates a very simple HTML form that contains three elements: the `Name` label, a text box that accepts the corresponding name, and the `Login` button. When you enter a name in the `Name` text box and press the `Login` button, the PHP script `cookies2.php` is called. This script, as explained earlier, then handles the further execution of the form.

Figure 24-12 shows the HTML form that is created using the above HTML code.

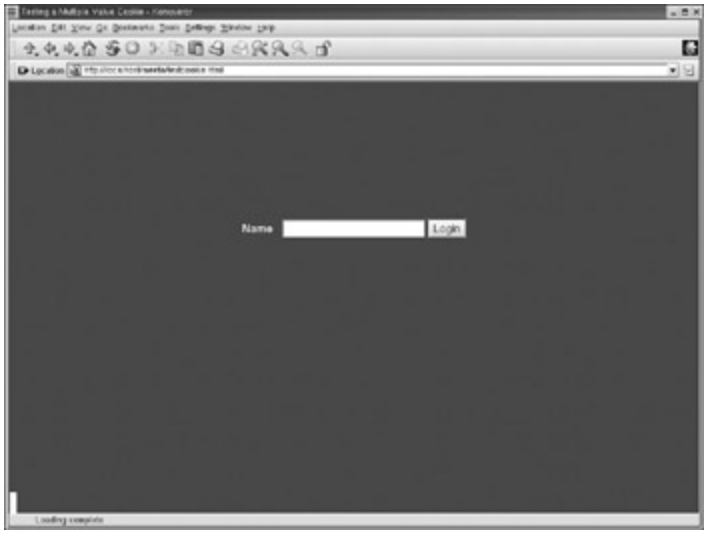

Figure 24-12: The HTML form that calls the cookie

This brings us to the final frontier of working with cookies, which is deleting cookies. In the next section, you'll learn about the same.

Deleting Cookies

After a cookie has served its purpose and you don't need it any longer, you might want to delete it. No use unnecessarily storing unwanted cookies! PHP does not boast of an exclusive function for this purpose. In fact, the good old `setcookie()` function once again comes to the rescue! Call the `setcookie()` function with only the name parameter when you want to delete a cookie. The syntax to delete a cookie is:

```
setcookie("cookie_name");
```

Therefore, if you would like to delete a cookie called `kookie`, you can delete it by using the following statement in your script:

```
setcookie("kookie");
```

Caution There is one very important fact about deleting cookies in PHP that you must know and remember when you delete one. If you would like to create a cookie with a name that already exists, you will first need to create it with a new value and only then delete the old cookie. This means if you need to create a cookie called `kookie` that already exists, you'll first write the following statement in your script:

```
setcookie("kookie", "New Cookie");
```

Only after you have reset the value of the existing cookie will you have the statement pertaining to the deletion of the cookie in your PHP script, which is:

```
setcookie("kookie");.
```

Baffling, isn't it? Let me explain why it is so. The reason behind this "anti-logic" rule is that all the cookies in a PHP script are sent to the Web server in a reverse order. This is the reason why you'll need a delete statement after the reset statement. If you forget to remember this simple fact, you might end up with so many errors and so frustrated that you'll never want to work with cookies in PHP anymore!

The common phobia among frequent Internet users and even some Web developers is that the use of cookies is dangerous! Is that so? Look for the answers in the next section.

Chapter 11: Handling Files

Overview

A well-designed Web site is a sight for sore eyes. However, the same Web site might lose its attractiveness if it was not up-to-date. If you have taken the trouble to put up a Web site, make a little more effort and keep the site updated! This holds especially true for the promotional or e-commerce sites. For example, new products are added to a vendor's inventory year round. Similarly, the rates and seasonal discounts are also subject to change frequently. However, if you make these changes to the embedded HTML code of the site, you are giving yourself a lot of unnecessary extra work. The simple remedy for this situation is to store frequently changing data in an external file and retrieve it from the same file. Then, instead of updating a changed discount rate manually in countless places, you can update it once in this related file. Your life is simpler! In addition, more often than not, this persistent data might prove to be important for later reference.

Data from a Web page can be permanently stored in two ways. You can either store the data in a file or store it in a database. In fact, a database is actually an advanced and more structured form of a file.

This chapter introduces you to the basics of handling files in PHP. You will learn about various file-related functions that allow you to access data from a PHP-based Web page, store data generated by a PHP page into a text file, or simply manipulate files while reading or writing to the file. You'll learn about storing data in a database in detail in Chapters 11 and 12.

Working with Files

If you have previous experience of working with other programming languages, such as C or C++, you might remember that working with a file can be broken down into the following stages:

- **Opening a file.** The specified file is opened. If the file doesn't exist, the file is created.
- **Manipulating the file.** After the file has been accessed, it is ready for manipulation. You can read from the file if it already consists of data, write to the file anew, or append data to the preexisting data in the file.
- **Closing the file.** After you have finished working with the file, you then need to close the file.

PHP offers you several file-related functions that you can use to save the data from a Web page to a file, retrieve the data from a file to a Web page, or simply change or append new data to an existing file. Some of the most frequently used file operations include the following:

- Checking for the existence of a file
- Opening a file
- Reading from a file
- Writing to a file

The next sections discuss these file operations in detail.

Checking whether a File Exists

Before you perform a file-related operation with the specified file, you might want to check whether the file already exists or not. You can use the `file_exists()` function to do so. The `file_exists()` function returns True (or `1`) if the specified file exists and False (or `0`) if it does not. The syntax of the `file_exists()` function is given below:

```
bool file_exists(string file_name);
```

As indicated in the above syntax, the `file_exists()` function takes only one argument—file_name, which specifies the name of the file whose existence is being tested.

Consider the following code to understand the `file_exists()` function.

```
<?php

   if (!(file_exists("data.dat")))
   {
    echo "The file exists.";
   }
   else
   {
    echo "The file  does not exist.";
   }
```

In the above code, the existence of the file `data.dat` is being verified with the help of the statement `if (file_exists("data.dat"))` . As you learned in Chapter 3 , "Variables, Operators, and Constants ", the `!` operator negates the result of the `if (file_exists("data.dat"))` statement.

Opening a File

You need to use the `fopen()` function to open a file in PHP. This function can open either a file or a URL. As a result, the file can be either a local file or a remote file. The `fopen()` function requires at least two parameters. The syntax for the `fopen()` function is given below:

```
int fopen(string file_name, string mode, int include_path);
```

> **Note** If the specified file needs to be retrieved from a remote HTTP server, remember that it can be opened only for reading. This effectively stops people from messing around with others' Web pages. Similarly, if the specified file needs to be retrieved from an FTP server, you must remember that you can open files for either writing or reading, but not both simultaneously.

In the above syntax, the argument `file_name` specifies the name of the file to be opened. If the value of the `file_name` argument begins with `http://` , it implies that the file is located on a remote Web server and an HTTP 1.0 session needs to be established with the specified server to open the file. Similarly, if the value of the `file_name` argument begins with `ftp://` , the specified file needs to be retrieved from an FTP server after an FTP session is established with the specified server. If the value of the `file_name` argument starts with `php://stdin` , `php://stdout` , or `php://stderr` , the corresponding standard Input/Output stream is opened. However, if the value of this argument begins with none of the aforementioned prefixes, the specified file is considered local and PHP searches for it on the local hard disk.

The second argument—`mode` —indicates whether the file was opened for reading, writing, or appending. The `mode` argument can take one of the values indicated in Table 11-1 .

Table 11-1: Values of the mode Argument

Mode	Description
r	Opens the file for reading only.
r+	Opens the file for reading and writing and places the file pointer at the beginning of the file.
w	Opens the file for writing only. If the file consists of any data, the data will be lost. If the file does not exist, creates the file.
w+	Opens the file for reading as well as writing. If the file consists of any data, the data will be lost. If the file does not exist, creates the file.
a	Opens the file for appending data after the preexisting data. If the file does not exist, creates the file.
a+	Opens the file for reading as well as appending. If the file consists of any data, the data will be written to the end of file. If the file does not exist, creates the file.

> **Note** You might come across the mode b in few cases. This mode works only for platforms, such as Windows, that can differentiate between the text and binary formats. This mode is useless on platforms such as Unix.

The third argument—`include_path` —is optional and is used if you want to search in the path specified by this

argument.

The following code demonstrates the use of the `fopen()` function.

```
if (!(file_exists("data.dat")))
    {
    $fp = fopen("data.dat","w+");
   }
   else
   {
//-----------If file exists, then open it in the append mode----
    $fp = fopen("data.dat","a");
   }
```

Caution Be very careful when inputting backslashes (\) in the local file path and forward slashes (/)in the path to a remote file (http:// or ftp://) if you are working on the Windows platform. If you mix up the slashes, you might end up with a warning from PHP to the effect that the specified file path was not found!

The above code first checks to see if the file `data.dat` exists. If the file doesn't exist, a new file called `data.dat` is created with the `w+` mode. This implies that the file is ready to be read from as well as written to. On the other hand, if the statement `if (file_exists("data.dat"))` returns True—that is, if `data.dat` already exists—the file is opened in `write` mode, allowing you to add data to the file.

You can use @ in front of the `fopen()` function or any other system function in PHP—for example, `@fopen("data.dat","w+");` . The use of @ in the beginning of any system function would help you to suppress standard compiler warnings in PHP. However, you must take care to provide custom error messages, in case the specified user-defined functions fail.

Now that you have learned to open a file, the next section tells you how to close the file.

Closing a File

In order to close a file, you need to use the `fclose()` function. If the `fclose()` function closes a file successfully, it returns True. If it encounters a failure in the file close operation, the `fclose()` function returns False.

The syntax of the `fclose()` function is:

```
bool fclose(int file_pointer);
```

The `fclose()` function takes a single argument in the form of the file pointer, `file_pointer` , which references the file that needs to be closed. For the function to execute successfully, it is important that the `file_pointer` argument be valid and references a file that was opened by using the `fopen()` function.

Now that you have learned to close a file, the next section discusses how to read from a file.

Reading from a File

You need to use the `fread()` function to read from an external file in PHP. The syntax for the `fread()` function is:

```
string fread(int file_pointer, int length);
```

As you can see, the `fread()` function takes two arguments—`file_pointer` and `length` . The `file_pointer` argument references the specific location in a file that has to be read. The `length` argument

specifies the number of characters that have to be read from the specified location. The read operation continues until all the characters specified by the `length` argument are read. If the end of file (EOF) is reached before the specified length, the read characters would be returned before the read operation is terminated.

Consider the following code:

```
                    $fo = @fopen("C:\PHP\Myfiles\data1.dat",
r) or die("Could not locate the specified file!");
                    //-----The first 124 characters of the file,
data1.dat, are read
                    $fr = fread($fo, 124);
                   }
```

In the above code, a file `data1.dat` is opened for reading with the help of the `fread()` function. The `or die ("Could not locate the specified file!")` statement is used to print a custom message if the `fopen()` operation fails. The result—`1` or `0` —of the `fopen()` function is stored in the variable, `$fo` . This variable is then supplied to the `fread()` function as the reference to the file to be read.

The value `124` in the `fread()` function specifies that the first 124 characters of data1.dat will be read.

PHP provides a wide range of file functions besides `fread()` . The `fgets()` , `fgetc(),` and `feof()` functions are the most commonly used. The next section gives you more information about these functions. You'll first learn about the `filesize()` function in PHP.

The *filesize()* Function

The `filesize()` function returns the total size of the specified file. However, if the specified file doesn't exist, False (or `0`) is returned. The syntax of the function is given below:

```
int filesize(string file_name);
```

As shown in the above syntax, the `filesize()` function takes a single argument—the name of the file whose size is to be determined.

> **Note** Note that the `filesize()` function works only with local files.

The *feof()* Function

You can use the `feof()` function to determine if the entire content of the specified file has been traversed and the file pointer has reached the end of file (EOF) position of the file. This function returns True if the file pointer is currently at EOF of the file; otherwise, it returns False. However, if an error occurs, the `feof()` function returns True.

The `feof()` function takes a single argument in the form of the file pointer, as shown in the syntax of the function below:

```
Int feof(int file_pointer);
```

> **Note** Note that the file pointer must point to a file that was successfully opened by the `fopen()` function.

The *fgetc()* and *fgets()* Functions

The `fread()` function returns a string of specified length from the specific location in a file. What if you would like to read a single character? You can easily manipulate the `fread()` function to read the specified character by placing the pointer at the desired location and specifying the length argument as `1` . However, PHP gives you a much simpler way to do this than manipulating file pointers and string lengths! It offers the `fgetc()` function to do

so. This function gets a single character from the position where the file pointer is located. The syntax of the `fgetc()` function is:

```
string fgetc(int file_pointer);
```

Note If it encounters EOF, `fgetc()` returns False. Also, the file pointer must point to a file that was successfully opened by the `fopen()` function.

The `fgets()` function returns a line (string) from the location specified by the file pointer. However, unlike the `fread()` function, the length of the string returned by `fgets()` is one less than the length specified by the `length` argument. The syntax of the `fgets()` function is:

```
string fgets(int file_pointer, int length);
```

The read operation is terminated if any of the following conditions is met:

- `length` -1 bytes have been read
- Newline character, `\n` , is encountered

uEnd of the file, `EOF` , is encountered

Consider the following code, which rounds up all the information you have gathered about `filesize()` , `feof()` , `fgetc()` , and `fgets()` functions until now.

```
<?php
$file = "data1.dat";
//-----Opening the file data1.dat.
//-----Is displayed.
$fo =@fopen($file, "r") or die("Could not locate the specified
file! Please check If the file is valid.");
//-----Calculating the total size of the file - data1.dat.
$file_length = filesize($file);
echo "The total size of the file is: $file_length", "\n";
//---Reading the entire file row-wise. Each time a line ends,
newline character is met, the variable
//---$total_rows Is Incremented by 1.
while(!(feof($fo)))
{
      $tr = fgets($fo, $file_length);
      $total_rows = $total_rows + 1;
}
echo "Total number of lines in this file is: $total_rows ",
"\n";
//---Closing data1.dat for the next loop to execute successfully.
fclose($fo);
//---Opening data1.dat again, so that the file pointer is at the
beginning of the file.
$fo1 = @fopen($file, "r");
//---Reading the entire file character-wise. Each time a
character is read, the variable
//---$total_chars is Incremented by 1.
while(!(feof($fo1)))
{
      $tc = fgetc($fo1);
      $total_chars = $total_chars + 1;
```

```
                }
               echo "Total number of characters in this file is:
$total_chars ", "\n";
                 ?>
```

In the above code, the file, `data1.dat` is opened. If the open operation is not successful (if the file is not found in the specified location), the corresponding message `Could not locate the specified file! Please check if the file is valid.` is displayed. On the other hand, if the file-open operation is successful, the total size of the file is calculated using the `filesize()` function. The total size of the file is stored in the `$file_length` variable. Next, the entire file is read line-wise (until the newline character) with the help of the `fgets()` function. The variable `$total_rows` is incremented for every new line that is read. When the file pointer, `$fo` , reaches the `EOF` of `data1.dat` , the condition `while(!(feof($fo)))` returns False and the loop is terminated. A message, `Total number of lines in this file is: 3` , is displayed. The file is then closed. You need to close the file at this point because due to the previous loop, the file pointer has already reached the `EOF` ! For the next loop (which counts the total number of characters in `data1.dat`) to execute successfully, the file pointer must not be at `EOF` . After the file is closed, it is opened again, which places the pointer in the beginning of the file. The next loop is executed and the total number of characters in the file is calculated with the help of function `fgetc()` . After each character, the variable—`$total_chars` —is incremented. When the EOF of the file is reached, the loop condition returns False and the loop is terminated. A message `Total number of characters in this file is: 54` is displayed.

Figure 11-1 displays the output of the preceding code.

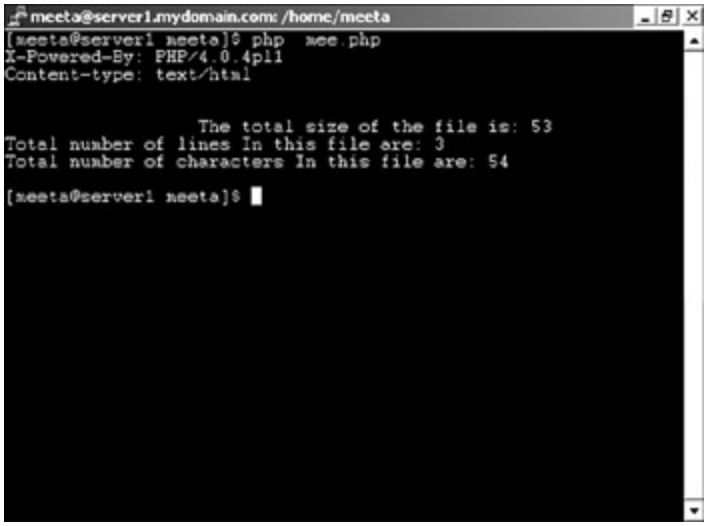

Figure 11-1: Output of the file functions

Figure 11-2 displays the content of the `data1.dat` file used in the preceding code.

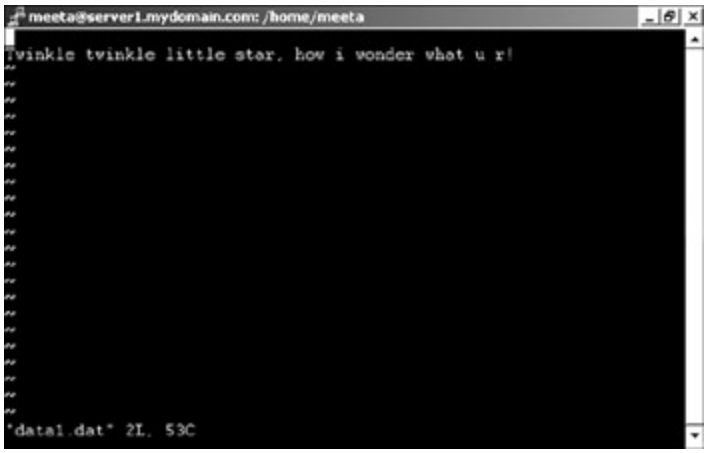

Figure 11-2: Content of the `data1.dat` file

In the next section , you'll learn about another function related to file-read operations, `fseek()` .

The *fseek()* Function

You can use the `fseek()` function to set the file pointer to a specific location with respect to the current position of the pointer. The syntax of the `fseek()` func- tion is:

```
int fseek(int file_pointer, int offset, int whence);
```

In the above syntax, the first argument is file pointer, `file_pointer` , which references the file where the desired operation is to be carried. The next argument, `offset` , is an integer that defines the starting point of the read operation in relation to the beginning of the specified file. The third argument, `whence` , specifies the end position in the file, which indicates where to stop reading the data. Both `offset` and `whence` are measured in bytes.

The `whence` argument has the following three standard values:

- **SEEK_SET .** Sets the file pointer at a position equal to the `offset` bytes.
- **SEEK_CUR .** Sets the file pointer at a position that is equal to the sum of the current location (byte) and the `offset` .
- **SEEK_END** . Sets the file pointer at a position that is obtained by the sum of EOF and `offset` .

Note SEEK_SET is the default value of the `whence` argument. So if the `whence` argument is not specified, the value of `whence` is assumed by PHP to be SEEK_SET .

To understand the `fseek()` function, consider the following code.

```
<?php
$file = "data1.dat";
//---Opening the file "data1.dat, else displaying a message
$fo =@fopen($file, "r") or die("Could not locate the specified
file! Please check if the file is valid.");
//---Determining the size of data1.dat
$file_length = filesize($file);
echo "\n";
echo "The size of the file is: $file_length.", "\n";
//---Moving the file pointer to 22nd byte in the file
$fs = fseek($fo, 22, SEEK_CUR);
```

```
                //---Reading five characters from the current position of the
file pointer
                $current_char = fread($fo, 6);
                echo "Currently, the file pointer is at $current_char in the
file.", "\n";
                ?>
```

In the preceding code, the file `data1.dat` is opened. If the file-open operation was unsuccessful, the message `Could not locate the specified file! Please check if the file is valid.` is displayed. On the other hand, if the `data1.dat` is opened successfully, the total size of the file is calculated with the help of `filesize()` function and displayed on the screen. Remember that because of the `fopen()` function that was used to open the file, the file pointer is located in the beginning of the file. The `fseek()` function is then used to move the file pointer `$fo` to the 22nd byte relative to the beginning of the file with the help of the argument `SEEK_CUR` . The `fread()` function is then used to read six characters from the new position of the file pointer. Finally, the six characters, `e star` , that have been extracted from `data.dat` are displayed.

Figure 11-3 shows the output of the preceding code.

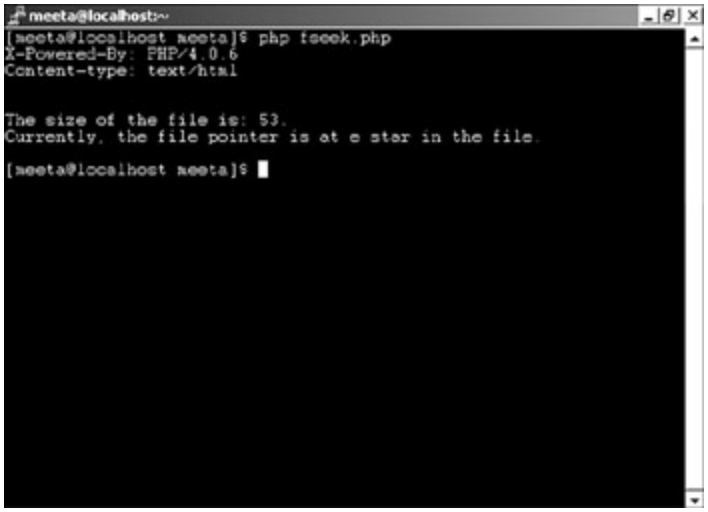

Figure 11-3: Content of the `fseek()` function in the preceding code

Now that you know how to read data from a file and various functions that help you to manipulate the read-related file operations, the following section discusses how you can write to a file.

Writing to a File

PHP provides the `fwrite()` function to write to an external file. This function takes three arguments. The syntax of the `fwrite()` function is given below.

```
int fwrite(int file_pointer, string string, int length)
```

The first argument, `file_pointer` , is the integer-type file pointer that references the file where you need to write. The second argument, `string` , is the information that needs to be written to the file. Finally, the third argument, `length` , is an optional argument. If it is specified, the write operation continues until the number of bytes specified by `length` have been written or the end of the string being written is reached, whichever is achieved first.

To better understand the `fwrite()` function, consider the following segment of a code.

```
$dataformat="A20A200A50A6A10A2A4A1A1A1A1A1A20A20A20A20A20A20A1";
                    $line =
                   pack($dataformat,$name,$address,$email,$gender,$birth_month,
                   $birth_day,$birth_year,$fiction,$action,$horror,$thrillers,
                   $comedy,$hobbies[0],hobbies[1],$hobbies[2],$hobbies[3],
                   $hobbies[4],$hobbies[5],"\n");
                    echo $line,"<br>",strlen($line);
                    $fp = @fopen("data.dat","w+") or die("Could not open the file.
Please check if the
                     file exists.");
                   //----Writing to the file
                   $result = @fwrite($fp,$line) or die("Data could not be written
to the file.");
                   fclose($fp);
```

In the above code, the file `data.dat` is opened in `w+` mode, which enables the file to be read as well as written to. If for any reason the `data.dat` cannot be opened, the message `Could not open the file. Please check if the file exists.` will be displayed. After the file is opened successfully, the string stored in the `$line` variable is written into the file. However, if the write operation was not successful, the message `Data could not be written to the file.` is displayed. The open file is then closed with the help of the `fclose()` function.

> **Note** Note that since `data.dat` is opened in the `w+` mode, if it contained some data, the data will be deleted and the new data (stored in the `$line` variable) will be written to the file.

The function `fputs()` works very much like the `fwrite()` function in PHP. In fact, `fputs()` is an alias of `fwrite()` . Therefore, the syntax of `fputs()` is also identical to `fwrite()` .

```
int fputs(int file_pointer, string string, int length)
```

Confused by the statement `$dataformat="A20A200A50A6A10A2A4A1A1A1A1A1 A20A20A20A20A20A20A1";` ? The next section explains what this is.

Formatting the Data While Accepting Input

Consider the following statement:

```
$dataformat="A20A200A50A6A10A2A4A1A1A1A1A1A20A20A20A20A20A20A1";
```

The preceding statement causes the input accepted from the HTML form that you designed in Chapter 8 for Project 1 (“Creating a User Registration Form for an Online Shopping Site”) into the variable `$dataformat` in a predefined format. The `$dataformat` variable stores the data related to a record in a single line, where each field entry is separated by fixed spaces reserved for them. The character `A` preceding a number (`20` , `200` , `50` , `6` , `10` , `2` , `4` , `1`) represents a variable, such as `$name` , `$address` , `$email` , and so on, in the order they were accepted from the HTML form. Therefore:

- **`A20`.** Reserves 20 spaces for the value of the corresponding variable. These include `$name` , `$hobbies[0]` , `$hobbies[1]` , `$hobbies[2]` , `$hobbies[3]` , `$hobbies[4]` , and `$hobbies[5].`
- **`A200`.** Reserves 200 spaces for the value of the corresponding variable. This single variable is `$address` .
- **`A4`.** Reserves 4 spaces for the value of the corresponding variable. This single variable is `$birth_year` .
- **`A50`.** Reserves 50 spaces for the value of the corresponding variable. This variable is `$email` .

- **A6 .** Reserves 6 spaces for the value of the corresponding variable. This variable is `$gender` .
- **A10 .** Reserves 10 spaces for the value of the corresponding variable. This single variable is `$birth_month` .
- **A2 .** Reserves 2 spaces for the value of the corresponding variable. This variable is `$birth_day` .
- **A1 .** Reserves 1 space each for the value of the corresponding variable. For each checkbox—`$fiction` , `$action` , `$horror` , `$thrillers` , `$comedy` —only one space has been reserved.

  Note The spaces reserved for each variable-value is fixed. If, for example, a user enters a name whose length is greater than 20, the characters from the 21st position will be truncated. Similarly, if a name entry is shorter than the stipulated 20 characters, extra spaces will be appended to the name until the name is 20 characters long.

You also saw the use of the `pack()` function in the preceding code. You'll learn more about the `pack()` and `unpack()` functions in the next section .

The *pack()* and *unpack()* Functions

You can use the `pack()` function to "pack" data in a string. The concept of this function was taken from Perl. The syntax of the `pack()` function is specified below:

```
string pack(string format, argument 1, argument 2,  ...  ... , argument n)
```

The `pack()` function formats the specified arguments—`argument 1, argument 2, ... ... , argument n` —as a string according to the `format` argument and returns the resultant string. To better understand the working of this function, consider the following code:

```
                    $line =
                  pack($dataformat,$name,$address,$email,$gender,$birth_month,
$birth_day,$birth_year,
                fiction,$action,$horror,$thrillers,$comedy,$hobbies[0],$hob
bies[1],$hobbies[2],
                  $hobbies[3],$hobbies[4],$hobbies[5],"\n");
                    echo $line,"<br>",strlen($line);
```

In the above code, the variables from `$name` to `$hobbies[5]` are formatted as specified by `$dataformat` . This implies that the registration information of a user, which was accepted from the registration form (`index.htm`), is presented in a single line, separated by spaces. The formatted string is then stored in the `$line` variable, which is then displayed on the screen with the help of the `echo $line,"<br>",strlen($line);` statement.

The `unpack()` function "unpacks" the data that you "packed" using the `pack()` function. In other words, the `unpack()` function extracts the data that was formatted using the `pack()` function and stores the extracted data of the unpacked string element-wise and transfers it to an associative array.

The syntax of the `unpack()` function is:

```
array unpack(string format, string data);
```

As you can see in the above syntax, the `unpack()` function takes two arguments—`format` and `data` . The `format` argument specifies the format according to which the string contained by the `data` argument will be formatted. Consider the following statement:

```
$dataformat = "A20name/A200address/A50email/A6gender/A10birth_month
/A2birth_day/A4birth_year/  A1fiction/A1action/A1horror/A1thriller/A1comedy
```

```
/A20outdoorsports/A20adventuresports/ A20popmusic/A20rockmusic/A20aggressivemusic
/A20photography";
$data= unpack($dataformat, $line);
```

In the above statement (from `reader.php`), the formatted string contained by the `$line` variable is unpacked in the manner denoted by `$dataformat` in first statement of the above code.

To understand what is going on here, you'll have to understand the first statement thoroughly. This statement consists of a list of format codes separated by `/` . Consider the first format code—`A20name/` —in the statement. Here A represents the corresponding (string) value passed by the variable earlier while packing the string. Similarly, `20` represents the space reserved for the string, and `name` represents the first element of the associative array, `$data` . Similarly, each format code in the first statement of the above example represents a particular variable, space reserved for it, and the element of the associative array, `$data` .

Now that you understand how the string contained by the `$line` variable will be de-formatted, understanding the working of the `unpack()` function should not be difficult anymore. The function reads the first 20 bytes of the string and assigns it to `$data["name"]` . Then it reads the next 200 bytes of the string and assigns it to `$data["address"]` . This continues until the last format code is unpacked and stored in the array.

You have gathered sufficient information to put what you learned earlier in this chapter into practice. Now it's time to do just that!

UNIT - 4
HANDLING DATA STORAGE, MYSQL DATABASE PROGRAMMING

Chapter 12: Handling Data Storage

Overview

In the previous projects, you learned to create a Web page using HTML and to accept information from a user in this Web-based form. This information was then stored in a text file. You also learned to retrieve information from this text file.

In this project, you will learn to store the user information in a database. In this chapter, you will learn about the basics of databases. In particular, you will discuss MySQL because it is the most commonly used database in PHP. This chapter will cover concepts like creating databases and tables. You will also learn to add, modify, and delete records in a database. In the next chapter, you will learn to store the user information accepted from an HTML form in a database instead of in a simple text file.

Before you learn how to create a database and link it to a Web page, let us discuss what is a database.

An Introduction to Database Concepts

Since the dawn of time, man has felt the need to store information. For example, the ancient Egyptians used logs to keep an account of the grain in their granaries. These logs are now considered to be the first form of flat files. In the previous chapter, you learned how to save the entered user information in a text file, which, in database jargon, is called a *flat file*.

A flat file stores all the information in a single file and you cannot connect one piece of information with the other. A flat file is mostly created in the form of a simple text file. All the text in a text file is stored without any formatting and as single lines of information of equal length.

Another way of storing text in a flat file is by separating each piece of information by using a comma. All the information relating to a specific type is grouped together in a single line. An advantage of using a flat file is that it takes less amount of space as compared to a database.

A flat file is a compromise between cost and features. It provides adequate efficiency and is easy to write. This is why programmers prefer using a flat file when the data only needs to be stored and cost is a major constraint. However, if you want to store data efficiently and retrieve it quickly in the manner you prefer, a database is the answer!

A *database* is used to store a large quantity of information. Unlike a flat files, a database stores information in a structured format and you can easily retrieve specific information from it. This information can be in the form of text, date, or graphics. An advantage of a database over a flat file is that you can extract specific information from it, which is not possible in case of a flat file.

A simple example of a database is a telephone directory. All the information in a directory, such as the names, addresses, and telephone numbers, are stored in a structured format. For example, you might have noticed that all the names in a directory are stored alphabetically. This helps in searching for specific information. If you need to search for the telephone number of a person called Hanks, Adrian, you can directly begin searching under the section that contains names beginning with H instead of starting from the beginning.

The information in a database is stored in the form of tables, fields, and records that play an important role in structuring the information as well as making searches fast and efficient. In the next section, you will learn about the concept of tables and their constituents in detail.

Tables

Most of you would have seen filing cabinets in offices. These cabinets are used to store paper files. Each cabinet, or at least each shelf of a cabinet, contains files of a similar type or category. For example, all the files related to sales might be stored in one cabinet and those related to production might be stored in another cabinet. Similarly, a database contains many tables and each table maintains data about a specific object or entity. For example, an Employee table would contain personal information about all the employees of a company. Similarly, a Customer table would contain details about customers, and a table called Sales would contain the company's sales information. Just as each cabinet has a label that helps you to identify the content of the cabinet or shelf without actually opening the cabinet, similarly, each table has a unique name that helps you to identify the table's content.

Let's take the example of a library database. The database contains three tables-`Books`, `Members`, and `Transactions`. The `Books` table contains information regarding all the books available in the library. The

`Members` table contains information regarding every member who has registered as a member of the library. Finally, the `Transactions` table contains information regarding every book that has been issued.

All the data in the database is stored in a specific format. Every table stores data in the form of rows and columns. A *row* stores data in a horizontal format, while a column stores data in a vertical format. The columns in a table are known as *fields*, and the rows are referred to as *records*. Figure 12-1 shows all the three tables in the database.

Books

Book_id	Name	Category	Price
B001	Let's talk about Computers	Business	100
B002	50 Mouthwatering Recipes	Cookery	125
B003	Living Life Without Fear	Psychology	150
B004	Chinese Cuisine, Anyone?	Cookery	200
B005	Stars Shine Down	Fiction	220

Members

Member_id	Name	Address	Phone_no
M001	Julie Andrews	4630 College Ave.	415834-2919
M002	Joan Allen	36 Broadway Ave.	415836-7128
M003	Anthony Willis	8 Silver Ct.	415986-7020
M004	Linda Lawrenze	674 Darwin Ln.	415848-2089
M005	Peter Holloway	36 Upland Hts.	301946-8854

Transactions

Tran_no	Issue Date	Return Date	Book_id	Member_id
T001	01/15/2002	02/15/2002	B001	M003
T002	01/16/2002	02/16/2002	B002	M002
T003	02/22/2002	02/22/2002	B003	M001
T004	02/22/2002	02/22/2002	B004	M002
T005	02/27/2002	02/27/2002	B005	M005

Figure 12-1: Tables in the Database

The following sections discuss the various constituents of a table in detail.

Fields

As explained earlier, fields are a collection of information of the same type or category. As you might recall, the library database consists of three tables - `Books`, `Members`, and `Transactions`. The `Books` table would contain fields like a unique identification of a book, the book's title, the category, and the price of the book. The `Members` table would contain details like a unique identification of a member, the member's name, the address, and the phone number. Finally, the `Transactions` table would contain a unique transaction number, the date on which a book was issued, the book's identification, the identification of the member to whom the book was issued, and the date the book is due for return.

Therefore, the fields of a table are nothing but the attributes that best explain the characteristics specific to the table. All this data is stored in separate columns and can contain both text as well as numeric data. However,

depending on your requirement, you can even restrict the type of data that can be stored in each field. For example, you can restrict the content of the field containing the members' phone number to only accept numeric values or the `Return date` field of the `Transactions` table to only contain dates. Just as in the case of a table, a field of a table also has a unique name called the *field name*, which you can use to reference each field separately.

You can also classify the fields in a table as either *required* or *optional*. As the name suggests, if the field is classified as required, a user will need to enter a value for the field. The user cannot leave it blank! For example, a field for a member's name cannot be left blank. However, if the field is optional, the user has the choice whether or not to enter a value for the field. For example, a member might not have a phone number. Figure 12-2 shows the `Members` table with the field names and values.

Note An *attribute* is a characteristic of an entity. For example, a name is a characteristic of an author and, therefore, it is an attribute of the author entity.

Figure 12-2: Fields in the Members Table

Note An *entity* is something that you can identify easily. For example, an entity can be defined as a person, an object, a place, an activity, or a concept about which you can store information. To reflect the real-world relationships, an entity can either be dependent on another entity or can be independent. You will learn about relationships and dependent entities in a later section called Relational Database Management Systems (RDBMS's).

You must have heard several times about records. No! We are not talking about musical or championship records. We are talking about the records in a database. Let's see what a record in a database is.

Records

Records are separate pieces of information about an entity. This information is grouped together and stored in a table. Each piece of information is stored in a separate field. For example, if the book, `Living Life Without Fear`, has been issued, then a corresponding record should be created in the `Transactions` table. Similarly, the details about the book would be a record in the `Books` table, and the details about the member would be a separate record in the `Members` table.

Figure 12-3 shows the records of the `Members` table.

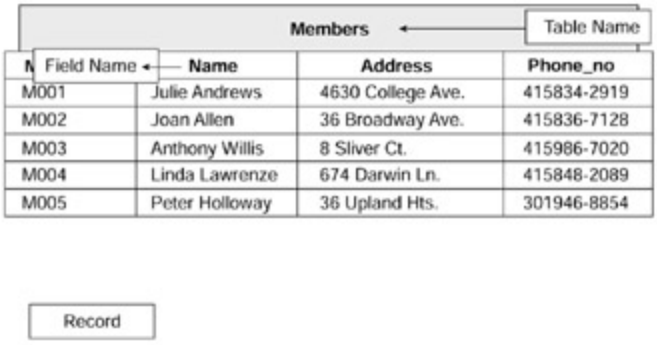

Figure 12-3: Records in the Members Table

You have just learned about databases. Next, you will look at the advantages of using databases.

Advantages of Using Databases

Using a database to store the information obtained by using a Web page has certain advantages. Some of these advantages are given below:

- Automation of the repetitive tasks performed while managing data
- Timely updating of data
- Provision of easy access to the data stored in various tables of a database
- Storage of a large volume of data in a systematic manner
- Quick and efficient search for and retrieval of specific data from a large volume of data

Database Management System

A *database management system* (DBMS) is a system by which all the data in a database is organized, stored, retrieved, and manipulated. A database contains a large quantity of information, which has to be stored in a central location, indexed, and stored in a sequential manner. A DBMS performs all these tasks. It also takes care of securing and maintaining the integrity of the data stored in the database. You can send instructions to the DBMS about the specific data you require and it retrieves the data and returns it back to you.

The concept of database management provides connectivity with software based on programming languages as old as C and COBOL. A DBMS provides as much flexibility as you require. For example, you can add new tables, fields, and records without disturbing the existing content of the database or the database structure.

A DBMS provides certain advantages such as:

- **Maintaining data integrity.** The use of a DBMS helps to maintain data integrity by not allowing the duplication of records where required or by ensuring that only one user has access to the data at one time.
- **Ensuring security of data.** A DBMS also ensures the security of data by allowing only authorized users to access or modify the database. Authorized users are provided passwords that they can use to access the database. A DBMS can also restrict the access of users to specific tables and fields in the database. For example, only an administrator might have the right to enter data about new books in the database and only the librarian might have the rights to issue books.
- **Querying for data.** A DBMS provides users the facility to search for specific information in a database. For example, you could search for all the books issued to a particular member or the total number of books issued on a particular day. Most DBMS's provide certain report writers to create these search strings, which are also known as queries.
- **Ensuring the independence of data.** While using a DBMS, you do not need to remember the structure of all the tables in the database every time you search for certain data. For example, while searching for a book issued to a particular member, you only need to send to the DBMS, the code equivalent of "give me the details of the member whose Member_id is this and who has been issued a book with this `Book_id` on the specified date." The DBMS searches for the record that matches the specified search criteria and sends it back to the calling program or the user.
- **Entering and updating data in the database.** Most DBMS's have a system that you can use to add or modify the data stored in a database. However, in this case, data entry is done in a specific format that you cannot change. If you need to enter data in a particular format or need to put certain restrictions on the type of the accepted data, it is advisable to create a program using a commonly used programming language such as C or Visual Basic for managing the data. You can then link the program to the database.

Relational Database Management System

As explained previously, a database has multiple tables and each table contains information of a specific type. Maintaining different tables helps in ensuring clarity in data storage and a faster retrieval of information from the database. You can retrieve different types of information from the tables by ensuring a relationship between them. For example, when retrieving the details of a book that is issued to a member, you could also retrieve information about the member who has issued the book and the transaction details. This is possible only if a relationship exists between the three tables: `Books`, `Members`, and `Transactions`. Adding fields from the other tables to a main table ensures a relationship between the different tables. For example, the `Transactions` table would contain the `Book_id` and `Member_id` fields. These fields form a link to the specific records in the `Books` and `Members` tables. In this manner, flexibility and speed in data retrieval is ensured.

You can have three types of relationships between the different tables in a database.

- **One-to-One.** In this case, exactly one record in both the tables map to each other. This type of relationship is not very common. It is used, for example, if a member is restricted to borrowing only one book at a time. However, this is an unrealistic scenario and, therefore, this type of relationship is not common.
- **One to Many.** This relationship implies that a reference to one record in a table is used multiple times in the other table. For example, in case of the `Books` and the `Transactions` tables, if a book can be rented out multiple times, you would have a single `Book_id` from the `Books` table being repeated multiple times in the `Transactions` table.
- **Many to Many.** This is the most common type of relationship, in which multiple records of a table map to multiple records of the other table. For example, it exists when a book can be borrowed by n number of members and a member can borrow n number of books.

You have just learned about the records in a table. However, you may wonder how you can decide which fields to include or not to include in a table. The answer is - through Normalization. Normalization is the process of determining what should be included in a table.

Database Normalization

Normalization is a process that is followed in a relational database management system to categorize data into smaller groups for easier and more efficient management. Although there are six stages in normalization, by the third stage, all the data in a table is dependent on only one key field. A key field is similar to a primary key that enforces the condition that only unique values can be stored in the field. This is what normalization is all about.

> **Note** You can create a relational database by using an RDBMS. Each table in a relational database is linked to another table by using a common attribute or attributes. For example, the `Books` table is linked to the `Transactions` table by using the common `Book_id` field. By linked, we mean that every `Book_id` that is inserted in the `Transactions` table should already exist in the `Books` tables.

Why We Need Normalization

The question that still comes to our mind is, “Why do we require normalization?” The answer is simple. Normalization helps us in reducing redundancy. If we have the same data repeated n number of times in our database, not only does the size of the database increase unnecessarily, there is also a problem with the storage and retrieval of data. Due to the large amount of data, a search for specific information not only takes a long time but also requires extra effort. This redundancy forces the following issues:

- **Inconsistencies.** Since data is entered repeatedly, there is bound to be inconsistency and, subsequently, the occurrence of errors.
- **Updating errors.** While inserting new records or modifying old ones, data can become inconsistent because you might update the information in one record and not in the others. Since the same information is repeated, the data will become inconsistent.
- **Linking errors.** If a relation exists between two tables, changing the values in one table can cause inconsistency in the data of the other table.

Normalization helps in changing the structure of tables so that they are easy to use, understand, and manage. In a normalized table, each record has a *primary key* and a group of attributes that can be defined as the characteristics of an entity.

Normalizing Forms

We mostly use the first four forms of normalization. These include the following:

- First normal form
- Second normal Form
- Third normal Form
- Boyce-Codd normal Form

The First Normal Form

A table is in the *first normal form* if each cell of the table contains only one value. In the first normal form, all the extra information has to be arranged properly. The first normal form is not found in relational databases since a relational database cannot contain an unnormalized table. Therefore, in a relational database, all the tables are already in the first normal form. For example, in Table 12-1, the data is not normalized because the `Projcode` field for the employee named `Jill` contains three values. After normalizing the table, the data would appear as shown in Table 12-2.

Table 12-1

Emp_id	Name	Projcode	Department
E001	Jack	P002	Acc
E002	Jim	P564	Mkt
E003	Mary	P888	Acc
E004	Jill	P564	Mgt
		P888	
		P002	
E005	Jones	P002	Mkt

Table 12-2

Emp_id	Name	Projcode	Department
E001	Jack	P002	Acc
E002	Jim	P564	Mkt
E003	Mary	P888	Acc
E004	Jill	P564	Mgt
E004	Jill	P888	Mgt
E004	Jill	P002	Mgt
E005	Jones	P002	Mkt

The Second Normal Form

In the *second normal form*, all the redundant information is removed and the information is stored in a separated table. For example, in the above code, since the `Projcode` and `Department` columns have redundant information (repetition of information about E004), you can create a separate table for it. Therefore, Table 12-2a is normalized to arrive at Table 12-3.

Table 12-2a

Emp_id	Name	Projcode
E001	Jack	P002
E002	Jim	P564

Emp_id	Name	Projcode
E003	Mary	P888
E004	Jill	P564
E004	Jill	P888
E004	Jill	P002
E005	Jones	P002

Table 12-3

Emp_id	Department
E001	Acc
E002	Mkt
E003	Acc
E004	Mgt
E005	Mkt

A table is in the second normal form if it is in the first normal form and all the attributes in the table are dependent on the main key or the primary key. In the above tables, the primary key is the `Emp_id`. In the above example, Table 12-3 is in second normal form because the records in the table are dependent on `Emp_id` and not just a part of the primary key. For example, `Department` is in no way related to the employee's name and project code.

You normalize a form to the second normal form by removing the attributes that are not fully related to the main key. You then group the removed information in another table. A common field connects both the tables to each other.

Third Normal Form

In the *third normal form*, the table must be in the second normal form and all the non-key fields (secondary fields, such as Department) are only dependent on the primary key, the `Emp_id`. In the above example, the project code is not dependent on the employee's name. Therefore, Table 12-2 can be further normalized into Table 12-4.

Table 12-4

Emp_id	Projcode
E001	P002
E002	P564
E003	P888
E004	P564
E004	P888

Emp_id	Projcode
E004	P002
E005	P002

Boyce-Codd Normal Form

A table is in a *Boyce-Codd normal form* if all the other keys are potential primary keys. Mostly, all the tables are considered to be normalized by the time you reach the third normal form. However, in certain cases, even if a table is in the third normal form, the table may not be fully normalized. A table is not normalized if:

- The values of two fields can be used together to get a unique value
- The values in two fields overlap and have at least one common attribute

If neither of the above conditions applies, then a table can be considered to be normalized in the third normal form.

Now that you know about the process of normalization, let's discuss the concept of denormalization in databases.

Denormalization

You just learned about normalization and how after normalizing, you obtain a set of tables that are related to each other and form a database. However, sometimes it is necessary to introduce redundancy in your tables. You must be puzzled! After trying to convince you of the virtues of having a normalized table where there is no redundancy, now I am telling you to introduce redundancy. No! This is not a mistake. Sometimes you need to sacrifice having a normalized database in favor of a database that provides faster query resolution. Having redundant data in tables helps in reducing the CPU time and disk I/O operation. Therefore, denormalization is the deliberate entering of redundant data in a table to improve performance.

Now that you know about databases, let's talk about the various databases supported by PHP.

Web Database Architecture

The time for static Web pages is long gone. Nowadays, more and more customers are asking for dynamic Web pages. The use of Web databases in Web page development requires developers to have an in-depth knowledge of both databases and the Web, since Web database application development is a synthesis of the two technologies.

The increased demand for Web-based databases can be attributed to an increase in the use of the Internet and intranet by the corporate world. This is fundamentally due to an increase in the use of the Client/Server technology in business applications. This technology is used by large organizations to provide all its employees simultaneous access to a database. This has introduced the concept of online databases in which a change made by one user is automatically reflected to the rest of the users. Previously, this technology used large data servers and client applications that connected to the main database. With advancement in technology, the industry has progressed to the use of Web technologies to ensure Client/Server connectivity. Since a browser is readily available, it is not necessary to create a separate application to provide a user interface. This technology is fast developing into the most efficient source of simultaneous database access.

There are many tools available in the market to create Web databases that not only provide methods for entering and modifying the data in databases but also methods for managing and delivering data to Web pages. It is important for developers who create Web databases and design queries to understand the fundamentals behind the designing of databases. An application that uses a Web database to store and retrieve data is known as a *Web Database application.*

A basic Web database application required the following components to function.

- **Database server.** The database server is used to store all the information entered by users and to display information on a Web page.
- **Web server.** The second component, the Web Server, contains the CGI and PHP scripts that are used to establish a connection with the database server.
- **Web browser.** The Web browser serves as the client interface. It is used to transfer information between a client and the Web server.

The HTTP protocol is used to transfer information from a Web browser to a Web server and vice versa. Similarly, ODBC and JDBC are some Application Programming Interfaces (API's) that are used to establish a connection between the Web server and the database. The above information is shown graphically in Figure 12-4.

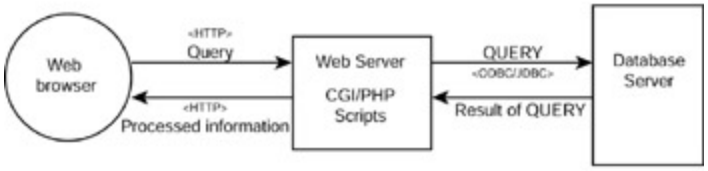

Figure 12-4: Web Database Application Architecture

Advantages and Disadvantages of Using Web Databases

The use of databases on the Web provides certain advantages and disadvantages. Listed below are some advantages of having a database-driven Web site.

- **Attractive and customized information.** You can customize the way information is available on your Web site. You can also make use of Web elements to make the data appear attractive.
- **Easier maintenance of content.** It is easier to maintain data on a Web site.
- **Freshness of data.** It is easier to update data in databases than in Web pages. In the case of dynamically generated Web pages, only the content or information needs to be changed, whereas manually changing a Web page requires adding addition content as well as maintaining the design of the Web site in relation with other Web pages.
- **Ready availability of data.** Most organizations already have the necessary data available with them. They only need to put this data on the Web.
- **Easy visibility.** It is easier to view data on a Web page instead of in a database.

However, the use of databases on the Web also has certain disadvantages. These disadvantages are listed below.

- **Investment in development time.** The Web page has to be developed, which involves the investment of time and effort.
- **Web skill set requirements.** You need developers with a skill set that maps to both Web application development and database proficiency. They should be able to build and maintain Web sites on a regular basis as well as work with and maintain the database. The former skill is not required when you use databases.
- **Investment in hardware, software, and employees.** You need to invest in additional employees to create and maintain a database-driven Web site. This is an added expense because the employees need to process the required skills to manage databases.
- **Scope of the user.** The data that you display on your Web page might not fulfill the requirement of the users who access your Web site. The users might use complex queries and get irrelevant records or no records at all.

You will now learn to create databases and tables in MySQL. You will also learn to insert, modify, and delete records in a table within a database.

MySQL Database Programming

There are many commands available in MySQL that you can use to perform database administration tasks such as creating databases and tables and viewing their structure. MySQL also provides commands for inserting, modifying, and deleting records from a database. Let's begin by understanding how to perform administrative tasks on a database.

Using the *mysqladmin* Command

Administering databases involves numerous tasks such as:

- Creating databases
- Deleting databases
- Checking the status of a database
- Resetting the values of the variables in a database

You use the `mysqladmin` command available in MySQL to perform these tasks. The syntax of the command is as follows:

```
# mysqladmin [options] Command1, Command2...
```

Table 12-5 lists the different tasks that you can perform by using `mysqladmin`.

Table 12-5: Commands in MySQL

Command	Task
create <database name>	To create a database with the specified name
drop <database name>	To delete the specific database along with all its tables
extended-status	To display the extended status information from the server
flush-tables	To clear all the table information
flush-host	To clear all the cached information
flush-threads	To clear all the stored thread information
flush-logs	To clear all the logged information about the server
refresh	To clear all the table information and to refresh all the logs
ping	To check and monitor connectivity with the mysql daemon
variables	To print all the information about variables
status	To display the status information at the command prompt
version	To display the version information from the server
shutdown	To shut down the server

Command	Task
password	To change the password

In the next section, you will learn to use the MySQL monitor. You can use the MySQL monitor to perform many tasks such as creating databases and tables.

Using the MySQL Monitor

MySQL is an effective database server developed by MySQL AB. The source code of MySQL is freely available and can be downloaded for free from the site, `http://www.mysql.com`. The software is compatible with both Windows and Linux platforms.

MySQL is a relational database management system that provides the facility to manage databases. You can also modify the source code to meet your requirements. MySQL also provides the following features:

- Support for languages such as Perl, Python, and PHP
- Support for a thread-based memory allocation system (Therefore, it is quite fast)
- Support for fixed as well as variable length records
- Support for a host-based verification system that provides security through verifying passwords that are encrypted during transit
- Support for large databases
- Support for different types of field datatypes
- Support for UNIX sockets, TCP/IP sockets, and Named Pipes for providing connectivity

Although MySQL is installed along with Linux, you can also install it separately. The recommended method for novice users is by using the RPM (Red Hat Package Manager) file. You can test if MySQL has been installed properly by executing some basic commands. We will begin by invoking the MySQL monitor from the command prompt. For detailed information on installing MySQL with PHP, refer to Chapter 2, 'Installing and Configuring PHP.'

To start MySQL, enter the following command at the command prompt:

```
[root@localhost root] # mysql
```

Figure 12-5 shows the output of the above command.

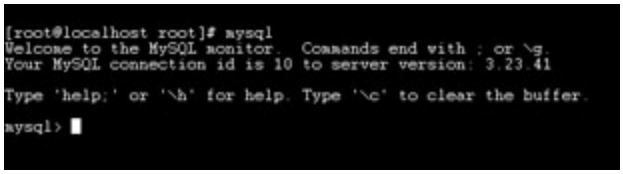

Figure 12-5: Using the MySQL monitor

Now that you have used the MySQL monitor, you can execute the MySQL commands for creating database and tables and for adding, modifying, and deleting records from the tables.

Creating Databases

In the previous section, you learned that you can create a database by using the `create <database>` command at the command prompt. However, you do not need to always execute this command from outside MySQL. You can also use the `create` command from inside MySQL. The syntax of the command is as follows:

```
mysql> create database if not exists Books;
```

Caution It is necessary to end all the MySQL commands with a semicolon (;) to indicate the conclusion of the command. Otherwise, MySQL keeps on waiting for a semicolon or returns an error.

The output of the above command is shown in Figure 12-6.

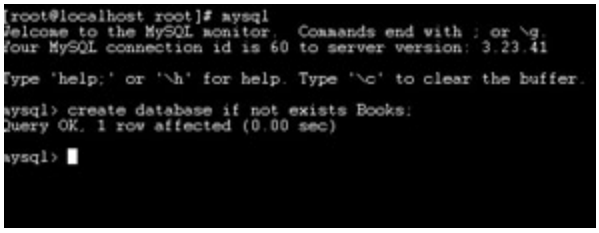

Figure 12-6: Creating a Database in MySQL

After a database has served its purpose and is no longer required, you should delete or drop it. This is important because if you do not delete the database, it occupies unnecessary disk space and is a burden on the system's memory. The syntax for dropping a database is given below.

```
mysql> drop database if exists Manuscript;
```

The above command will search for a database named `Books` and delete it. Another way of deleting a database is by using the `mysqladmin` command. The syntax of the command is given below.

```
mysql> exit;
 [root@localhost root] # mysqladmin drop Manuscript;
```

Caution The above commands assume that you have already created the `Manuscript` database.

While deleting or dropping a database, you will notice that you are asked to confirm whether you want to delete it. This ensures that you do not accidentally delete a database. You need to type y to complete the deletion. A message then appears confirming that the database has been deleted. The output of a sample deletion is shown in Figure 12-7.

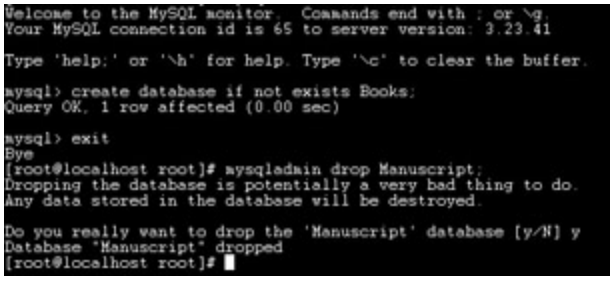

Figure 12-7: Deleting a Database in MySQL

In the next section, you'll learn to create tables in a database using MySQL.

Caution You must be cautious while deleting or dropping a database because once a database is deleted, its content cannot be retrieved.

Creating Tables

In the previous section, you learned to create and delete a database. In this section, you will learn to create a table and store data in it. You can create different tables to store different types of data. For example, you can create a `Books` table to store information about books and an `Authors` table to store information about authors.

The first step while creating a table is to obtain access to the database. Obtaining access implies gaining control of a database so that you can make modifications to it or create tables in it. The MySQL command is used to gain access to a database. After you have gained access to a database, you can use the different commands available in it to create tables and to view their structure.

Now, we will look at how we can perform these tasks.

Using a Database

As explained earlier, the first step in creating a table is to obtain access to the database in which you want to insert the table. You use the syntax given below to access a database. At the MySQL prompt, enter the code given below.

```
mysql> use Books
```

Caution You must ensure that the `Books` database already exists before you try the above command.

On the successful execution of the above code, you will be able to access the database and the following message appears: '`Database changed`". This message implies that the database named `Books` exists and is ready for use. The output of the above command is shown in Figure 12-8.

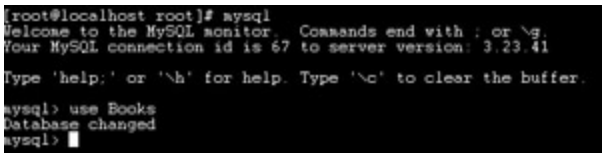

Figure 12-8: Accessing a Database

Now that you have gained access to the database, you can begin to construct tables for the database. However, before you go on to the actual creation of a table, you need to know about the different field types available in MySQL. Each field in a table needs to belong to a specific datatype. You will learn about these datatypes in the next section.

Field Datatypes in MySQL

As you already know, tables contain a set of fields that you can use to store values. The field types are the datatypes that determine the type of data that can be stored in each field. These field types and their description are listed in Table 12-6.

Table 12-6: Field Datatypes in MySQL

Type	Description

Type	Description
char	Used to store the string type data.
varchar	Used to store string values but of a larger size than a char datatype.
blob/text	Used to store strings of characters. The fields can store a maximum of 65535 characters.
int	Used to store integer type data. The maximum size of the datatype is 2147483646 numeric values.
bigint	Used to store integer values. This is similar to `int`, the only difference is that you can store a maximum of 9223372036854775806 numeric values.
smallint	Used to store small integer values. A smallint field can store a maximum of 32766 numeric values.
tinyint	Used to store integer values. This is the smallest of the integer datatypes and can store a maximum of 126 numeric values.
float	Used to store decimal values.
date	Used to store date values. The values are stored in the yyyy-mm-dd format and range from 1000-01-01 to 9999-12-31.
datetime	Used to store the time along with the date.
year	Used to store the year value.

Using the *create* Command to Construct Tables

You have already obtained access to the database named `Books`. Now you will learn to create new tables in the database. The syntax for creating a table is given below.

```
mysql> create tablename (first_fieldname fieldtype,
second_fieldname fieldtype,...,n_fieldname fieldtype);
```

You will now use the above syntax to create a table named `Products`, which has five fields. The fields and their datatypes are shown in the code given below.

```
                        create table Products (
                                    Productid bigint(20) NOT NULL
auto_increment,
                                    Name varchar(40) NOT NULL default ‘',
                                    Description varchar(200) NOT NULL default
‘',
                                    Price varchar(20) NOT NULL default ‘',
                                    Category varchar(20) NOT NULL default ‘',
                                    PRIMARY KEY (Productid)
                                  );
```

The preceding code creates a table named `Products`, which contains fields with the following characteristics.

- The table contains five fields: `Productid`, `Name`, `Description`, `Price`, and `Category`.
- The `Productid` field is of the *bigint* datatype and can contain a numeric value, with a maximum size of `20` characters. It also cannot contain a `NULL` value and the value is auto incremental in nature. This

means that every new record takes the previous record's `Productid` and adds one to it, to create the current record's `Productid` value.

- The `name` field is also of the *varchar* datatype and contains a string with a maximum of `40` characters. The content of the filed cannot be `NULL`.
- The `Description` field is of the *varchar* datatype and has a maximum size of `200` characters. This field also cannot contain a `NULL` value.
- The `Price` field is of the *varchar* datatype and can contain a maximum of `20` characters.
- The `Category` field will contain the category to which the product belongs. This field also belongs to the *varchar* datatype and can contain a maximum of `20` characters.
- The primary key for the table is `Productid`. This means that this field will have unique values and the records in the table are indexed based on these values.

The output of the above code is given in Figure 12-9. The figure shows the confirmation message that the table has been created.

Figure 12-9: Table Created in MySQL

Now that you have learnt to create a table, you will learn to review the contents of a database and lists the tables that it contains.

Viewing the Table in a Database

While you are working with a database, you might not remember the different tables you have created in it. You might also not remember the name of a specific table in the database. MySQL provides a command that you can use to list the various tables in the database. In the current example, you will display the table you just created in `Books`.

MySQL provides the `show tables` command to display all the tables available in the currently open database. The syntax of the command is given below.

```
mysql> show tables;
```

The output of the above command is shown in Figure 12-10.

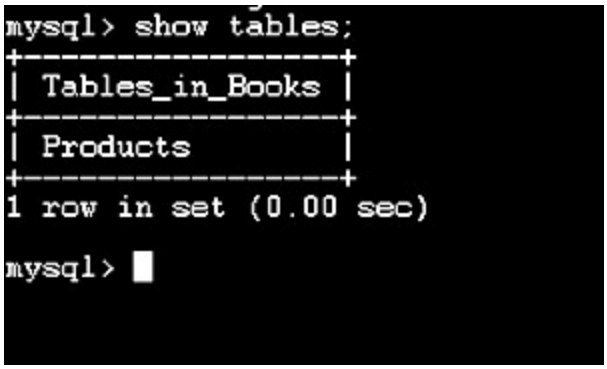

Figure 12-10: Previewing Tables in MySQL

Suppose you are working with multiple tables. How will you keep track of which table contains which fields? In the next section, you will learn how you can review the structure of a table.

Viewing the Table Structure

You might also want to find out the fields that are available in a table. MySQL provides a command to view the structure of a table. The command to view the fields available in a table is given below.

```
mysql> explain Products;
```

You can use this command to find out the fields of a table (`Products`, in the above case) and their datatypes. This information is important while designing queries because unless you know the fields that are available in a table, you cannot create effective queries. The output of the above command is shown in Figure 12-11.

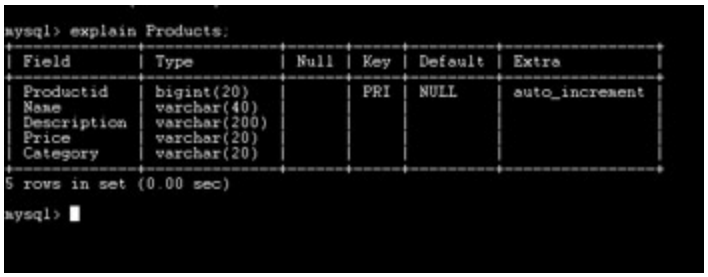

Figure 12-11: Viewing the Table Structure

As you can see, the output contains six columns. The names of these columns and their explanations are given below.

- **Field.** This column contains the names of the fields that were specified while creating the table.
- **Type.** This column contains the datatypes for each field. As explained previously, the datatype can be *text*, *char*, *integer*, *varchar*, *float*, or any one of the datatypes listed in Table 12-2. These datatypes are specified at the time of creating the table. Once a datatype has been specified, the field can only contain data that belongs to the specific datatype. For example, if a field is of the *char* datatype, it will store text as characters or strings even if the text contains numbers. The maximum size of data that the field can store also appears within parentheses along with the datatype.
- **NULL.** This column determines if a field can contain a blank value. If a field is blank, it will contain a `NULL` value.
- **Key.** This column specifies which field is the primary key. A field that contains the primary key can contain

only unique values, which means that you cannot repeat the same value for any other record. This also helps in indexing the values in the field. There can be a maximum of one primary key in each table and none of the values in the field can be `NULL`.

- **Default.** The column specifies a default value for a field. You can set a default value for a field to be used in case the field is left blank at the time of data insertion. This column is critical for fields that have been set to `Not NULL`.
- **Extra.** This column specifies any extra information about the field that MySQL can use while executing queries.

Now that you know how to create a table, you will now learn to can add, modify, and delete records in the table.

Entering Data into a Table

Now that you have successfully created a table, let's discuss how you can enter data into the table. You can use the `insert` command to enter data. The syntax of the `insert` command is given below.

```
                                mysql> insert into <tablename> values
(‘value_for_fieldone', ‘value_for_fieldtwo',
                                 ....'value_for_nthfield' );
```

Let's use the above syntax to insert new records in the table.

```
                                mysql> insert into Products values
(‘', ‘Great Expectations', ‘Written by Charles
                                 Dickens','200', ‘Classic');
```

The output of the above command is given in Figure 12-12.

Figure 12-12: Entering Records in a Table

You can similarly add other products to the `Products` table. Let's add another record to it.

```
                                mysql> insert into Products values
(‘', ‘Last of the Mohicans', ‘Written by Charles
                                 Dickens', ‘150', ‘Classic');
```

Now that you have learned to insert records in a table, you will learn to view the data entered in the table.

Viewing the Data in a Table

Once all the data has been entered into a table, you might want to review it. You can use the `select` command for this purpose. The `select` command is used to retrieve the records from a table or tables based on certain conditions or criteria. The syntax of the command is shown below.

```
                                mysql> select [fieldname] from
[tablename] where [expression] order by [fieldname];
```

The code given below retrieves all the records from the specified table. The output of the code is shown in Figure 12-13.

```
mysql> select * from Products;
```

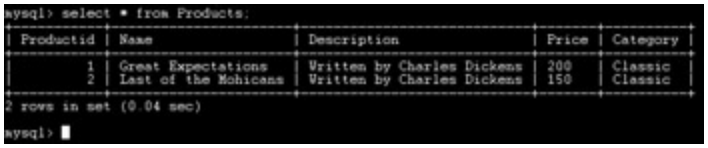

Figure 12-13: Viewing the Data in a Table

You can also restrict the output of the `select` command to specific fields by using the field names instead of an asterisk (`*`). For example, you can restrict the output to display only the `Productid`, `Name`, and `Price` of the product. The code to achieve this is given below, and the output appears as shown in Figure 12-14.

```
mysql> select Productid, Name, Price
from Products;
```

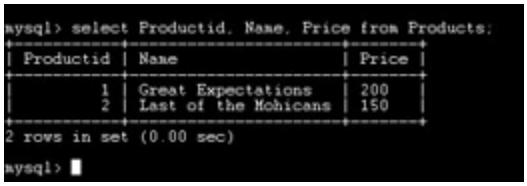

Figure 12-14: Viewing Selective Fields

You can also restrict the output of the select command by using the `where` condition. In the next section, you will learn to use complex select statements to retrieve data based on specific conditions.

Complex *select* Statements

Now that you know the basic format of the `select` command, you will see how to use the `select` statement to create complex queries. As explained previously, you use the `where` clause if you need to search for specific information in the database. An example is given below.

```
mysql > select * from Products where Price
=200;
```

The above code will display all the records in the `Products` table whose price is equal to `200`. The output for the above code appears in Figure 12-15. You can further qualify the `where` statement by using the comparison operators that you learned about in a previous chapter. Some of these comparison operators are shown in Table 12-7.

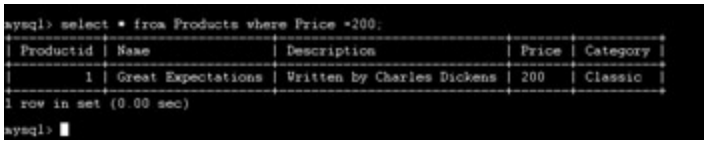

Figure 12-15: Viewing Selective Fields Based on a Condition

Table 12-7: Comparison Operators

Operator	Purpose
==	Equal to

Operator	Purpose
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to
like	Compares string text

Note The `select` command is a very powerful tool. You can use it to create queries that search multiple tables based on a criterion and return a specific output.

You can also use the `all` and `distinct` keywords to select all or specific records from the database. The `distinct` keyword will return specific records from specific columns. It also ignores duplicate records retrieved from the columns specified in the `select` statement. An example is:

```
mysql >select distinct Category from Products;
```

The above code will return all the records containing unique categories from the `Products` table.

You can use the `And` and `or` operators to combine two or more conditions. The difference in the use of both the operators is their implementation. In case of the `and` operator, all the conditions need to be fulfilled for the record to be included in the output, whereas in the case of the `or` operator, even if one of the conditions returns True, the record is included in the output. In the example given below, the output will contain all the `Productid's` whose `Category` is `Fiction` or whose `Price` ranges between `100` and `150`.

```
mysql > select Productid from Products where
Category = ‘Fiction'OR Price >100 AND
Price<150;
```

Another clause that can be used along with the `select` statement is the `group by` clause. You use this clause to group the records containing similar data in specific columns. Once the data is grouped, you can use aggregate functions such as `sum` and `max` to perform calculations on this data. For example, in the code given below, all the records are grouped together based on the `Category` field and the maximum price in each `Category` is displayed.

```
mysql > max(Price), Product_id, Name from Products
GROUP BY Category;
```

Having learned about inserting records, in the next section, you will learn to modify the existing records in a table.

Modifying the Data in a Table

You can use the `update` command to make changes to the data stored in a table. You can make changes to a single record or to multiple records in a database. A point that you need to remember is that the `update` command does not return a set of records, it only makes changes to the data in the database. The syntax of the `update` command is given below.

```
mysql> update <tablename> set
<fieldname> = ‘<New value>' ;
```

For example, you can change the value for the `Category` field of all the records to `Fiction`. The code to

perform the task is given below and the output is shown in Figure 12-16.

```
mysql> update Products set Category =
‘Fiction';
mysql> select * from Products;
```

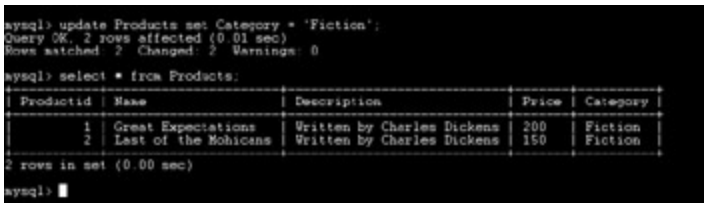

Figure 12-16: Modifying the Data of a Specific Field

You can also use the `where` clause to narrow down the update based on a specific condition. For example, you may want to increase the price of the products whose value is more than `150` to `200`. The code to do so is given below.

```
mysql> update Products set Price = ‘200' where
Price > 150;
```

You can also use the `update` statement to modify records in multiple tables based on a condition. This type of updating is called `cascade updation` since the changes made in one table are reflected in all the related tables.

After having learned how to insert and modify the records in a table, you must be curious about to delete records from a table. The next section covers this aspect.

Deleting Data from a Table

You can delete information from a table by using the `delete` command. You can delete single or multiple records from a table by using this command. You can also use the `where` clause with the `delete` command to delete records based on certain conditions.

> **Caution** Just as you need to be cautious while deleting a database, you also need to be cautious while deleting records. This is because once a record is deleted, it cannot be recovered.

The syntax of the command for deleting a record is given below.

```
mysql> delete <fieldname> from
<tablename> [where expression];
```

You can use the code given below to delete the records of all the products whose price is below `200`.

```
mysql> delete from Products where
Price<200;
```

The output of the above code is shown in Figure 12-17.

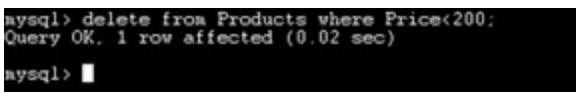

Figure 12-17: Deleting a Record Based on a Specific Condition

> **Caution** If you do not specify a field name, then all the records in the table will be deleted.

You must now be confident about adding, modifying, and deleting records from a table. However, suppose you need to modify the structure of a table. The next section explains this concept in detail.

Modifying the Table Structure

You can modify the structure of a table by using the `alter` command. You can perform the following tasks by using the `alter` command.

- Adding a column to a table

```
mysql> alter table <tablename> add <new field> <definition> ;
```

- Changing the datatype of a table

```
mysql> alter table <tablename> change <columnname> <newdefinition>;
```

- Indexing a table

```
mysql> alter table <tablename> add index <columnname> (<columnname>) ;
```

- Adding a unique column to a table

```
mysql> alter table <tablename> add unique <columnname> (<columnname>) ;
```

- Deleting a column from a table

```
mysql> alter table <tablename> drop <columnname>;
```

By now, you must be comfortable with using databases and tables. In the next chapter, we will look at how we can link a Web page to a database and store information in the database.

UNIT - 5
USING PHP WITH SQL DATABASE

Chapter 13: Using PHP with SQL Databases (MySQL)

In the previous chapter you learned about the basic concepts of MySQL. You learned about databases and how to cr them. You also learned about tables, fields, and records that form the basics of databases. In this chapter, you will le link the code written in PHP with a database. You will also learn to insert the records in the database and retrieve the records. In the previous chapter you learned to perform the above actions in MySQL; now you will perform them in programs written in PHP. However, in PHP's case, you will use an HTML form to accept the information from the use send the query to the MySQL database.

Working with MySQL

Nowadays, the success or failure of a Web site does not depend entirely on the attractiveness of the Web page. The content or information that your Web page provides also determines the success or failure of your site. This is the re why a majority of organizations want dynamic Web pages. The use of Java and DHTML may facilitate the creation of dynamic Web pages, but using dynamic content in these Web pages is a different issue. With the rapid growth of the Internet, more and more users are visiting Web sites. These users expect the Web sites to always have the latest information. If the users do not find the latest information on a site, they normally do not return to the site. Therefore, constant updating of the Web site is required. The question that arises is who is responsible for changing the conten the Web site. The responsibility can rest neither with the people who provide the content for the Web site nor with the developers who create the HTML Web page. In addition, content providers may have absolutely no knowledge of HT Their responsibility is limited to providing content for the Web page. The Web page developers, on the other hand, a preoccupied with creating the Web page and are not expected to periodically transfer data between Word documents the Web page.

The periodic maintenance of the Web site poses another problem. Most organizations prefer to continue with only on design because changing the design may mean investing in creating a new Web page. This involves both time and m

A solution to these problems is using data-driven Web sites. In data-driven Web sites, the storage of content and the hosting of the Web site are done at separate locations. You have a simple HTML Web page that acts like a template different types of content. The content management system integrates both the content and the Web pages. In this c no separate HTML coding is required to update the content each time the content is changed.

As explained earlier, there are two components in use, the PHP script that accepts the query, and the database, whic this case is MySQL. You have already learned how you can embed a PHP script in an HTML Web page. In this chap you will learn how to combine the simple HTML Web page you learned to create in Chapter 9 with the database you learned to create in the previous chapter . You will also learn how you can insert records in a database by using Web pages. You will also learn to query the database from the Web pages and display the result.

You will now learn how to connect to the MySQL database server.

Connecting to a Database

The basic idea behind a database-driven Web site is that the information to be displayed on the Web site is stored o database server. This information is then extracted dynamically from the database and displayed on the Web page. T dynamic Web pages can then be viewed in any ordinary Web browser. The process of connecting your Web site to a database is explained below.

The users enter the address of the Web page in their Web browsers. This information is sent to the Web server that

passes the information to the database server in the form of a query. The database server then returns the result bac the Web server that formats the information into an HTML format and displays it in the browser.

However, before you can establish a connection between the MySQL database and the Web page, you must learn to connect to MySQL. PHP provides a built-in function to make this connection. This function is called the `mysql_connect()` function.

You use the `mysql_connect()` function to establish a connection with the MySQL database. This function also acc certain parameters that determine who can connect to the database and where to connect. The function returns a va that confirms that the connection has been established. This value is stored in a variable known as the connection identifier. This identifier is used internally by PHP to verify that the connection has been established. If multiple connections are made to the same database by using the `mysql_connect()` function, only the first instance of the function actually establishes the connection. All the subsequent connections use the connection identifier returned by first command. The syntax of the `mysql_connect()` function is given below.

```
$connect = mysql_connect(<address>, <user id>,
<password>);
```

The command can contain three parameters. The explanations about these parameters and the connection identifier as follow.

- **Address.** The `address` parameter contains the IP address or the host name of the computer where the MySQL server is installed. You can specify the address as `localhost` if MySQL is running on the same computer as th Web server.
- **User id.** The `user id` parameter contains the user ID of a user who has rights to access the database. This us should exist on the MySQL server and should have been assigned the appropriate level of rights.
- **Password.** The `password` parameter provides the corresponding password to the ID specified in the user ID parameter. A password is used to ensure that only an authorized person is given access to the database. When parameter is not provided, the password is assumed to be blank.
- **$Connect.** The `$connect` variable is the connection identifier. This variable contains the value True if the conn is established. It contains the value False if the connection is not established. You can use the variable in your c refer to the connection or connect to the database.

```
<?php
        $connection= mysql_connect("localhost","root","")
or die("A connection to the Server could not be established !");

echo "Root user login in MySQL server  @ localhost successful.";
  ?>
```

In the code specified above, a connection is established with the MySQL server by using the user ID `root` . The add is `localhost` because both the Web server and the MySQL server are running on the same machine. The confirma of whether the connection is successfully established is stored in the variable `$connection` . If the connection fails, program execution ends after displaying an error message. You use the `die()` function to display this message. The `die()` function is one of the error handlers available in PHP. You can also use the @ symbol to suppress the defaul errors displayed by the system and instead display customized messages created by you. You will learn about error handlers and how to create customized error handlers later in this chapter. The output of the code given above appe Figure 13-1 .

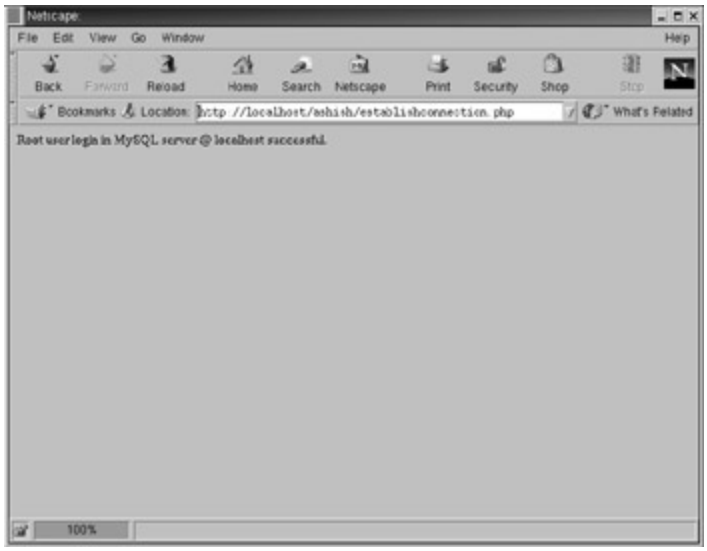

Figure 13-1: Connecting to MySQL

Now that you have established a connection with the MySQL server, you need to select the database in which you w insert or modify data. Suppose you do not have a database. In such a case, you first need to create the database. Th following section covers this procedure.

Creating a Database in MySQL

The `mysql_create_db()` function is used to create databases in MySQL. The following is the syntax for the comm

```
$createtable = mysql_create_db($dbname);
```

The `$dbname` variable contains the name of the database to be created, and the `$createtable` variable contains t result of the attempt to establish the connection. The result can be either True or False depending on whether the connection is successful. The code given below establishes a connection with a database named `Mytestdb` .

```
<?php

   $connect = mysql_connect("localhost","root","")
                                          or
die("A connection to the server could not be established !");

$createdb =mysql_create_db("Mytestdb")
   or die("database could not be created !");

echo "Database Mytestdb was created successfully.";

      ?>
```

In the code given above, a connection is established with MySQL by using the `mysql_connect()` function. Next yo create a database named `Mytestdb` on the MySQL server by using the `mysql_create_db()` function. The confirr whether the database has been successfully created is stored in the variable `$createdb` . If the database is created successfully, the value is True. Otherwise, it is False. The output of the code given above appears in Figure 13-2 .

Figure 13-2: Creating a database in MySQL

You will now select the database in which you want to insert or modify information.

Selecting a Database

Once you have created a database through PHP, or if the database already exists in MySQL, you can add tables and insert records in the database. Select a database before you begin to work with it. PHP provides the `mysql_select_db()` function to select a database in MySQL. Following is the syntax of the function.

```
$connect = mysql_select_db($dbname)
```

In the syntax given above, the database name is passed as an argument to the function. The function passes a value the `$connect` variable, confirming that the database has been successfully selected. If the value is True, then the database has been successfully selected. Some of the reasons why the variable could contain False are:

- The database does not exist on the MySQL server.
- The database is locked for exclusive use by another developer. If multiple users try to access the same database simultaneously and if one user exclusively locks the database, then the other users will get an error message.

```
<?php

$connect= mysql_connect("localhost","root",
       or die("A connection to the server
could not be established !");

 $result=mysql_select_db("Mytestdb")
 or die("Database could not be selected");

echo "Database Mytestdb was
successfully selected.";

?>
```

In the code given above, you are connecting to the MySQL server by using the `mysql_connect()` function. After yo

establish the connection successfully, you select the database named `Mytestdb` by using the `mysql_select_db(` function. If the database is selected successfully, then the variable `$connect` contains the value True. If the variable contains the value False, the `die( )` function is called, which displays a customized error message and closes the W page. The output of the code specified above is given below in Figure 13-3 .

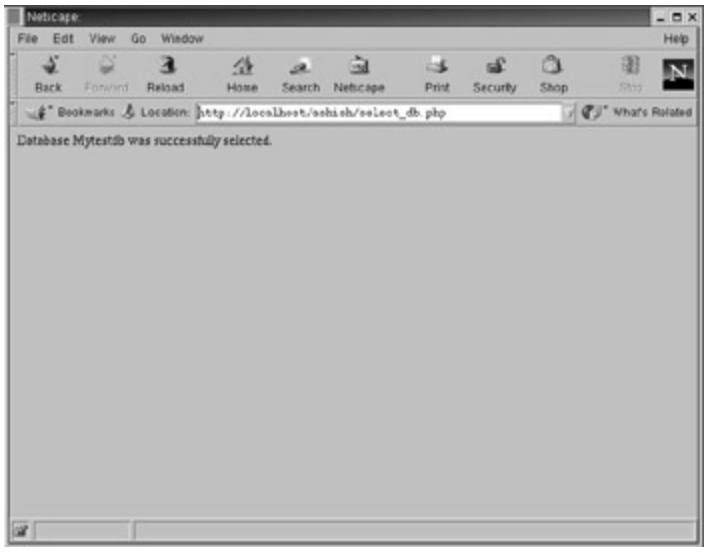

Figure 13-3: Selecting a database in MySQL

In the next section , you will learn how to create a new table in the database. You need to create a table before you c begin to insert information in the database.

Creating a Table in a Database

In Chapter 9 , you learned to save in a simple text file the information entered by a user. You did not use any tables; saved the information directly into the text file. In the case of databases, this is not possible. You need to create table before you can store information in the databases. In the code given below, you will create a table named Maildata. L you will store records in this table. In Chapter 11 , 'Handling Files ,' you learned about tables, fields, and records.

```
<?php

$connection=
mysql_connect("localhost","root","")
or die("Could not connect to MySQL in localhost

$selectdb=mysql_select_db("Mailinglist")
or die("Could not select Mailinglist Database!"

$sqlquery = "create table if not exists Maildata
(Name varchar(50) Not
Null, Email varchar(50) Not Null, Secondaryemai
varchar(50) Not Null Primary
Key)";

$queryresult = mysql_query($sqlquery)
or die(" Query could not be executed.");
```

```
                                        echo " Table Maildata successfully created
in the Mailinglist database.";
                    ?>
```

After the connection with MySQL has been established and access to the Mailinglist database secured in the code gi above, you create a table named Maildata by using the `create table()` function. The table has three fields: `Name` `Email` , and `Secondaryemail.` The datatypes of all the fields are varchar and you can store a maximum of 50 characters in each of the fields. Neither of the fields accepts a NULL value, and the primary field in the table is `Emai` This means that

the fields will contain unique values. Therefore, two people sharing the same e-mail ID cannot log on to the database simultaneously. The output of whether the `select` statement is successful is returned to the `$queryresult` variab the query returns records, the variable will contain the value True, and if the variable contains the value False, an err message is displayed. The output of the code specified above is given in Figure 13-4 .

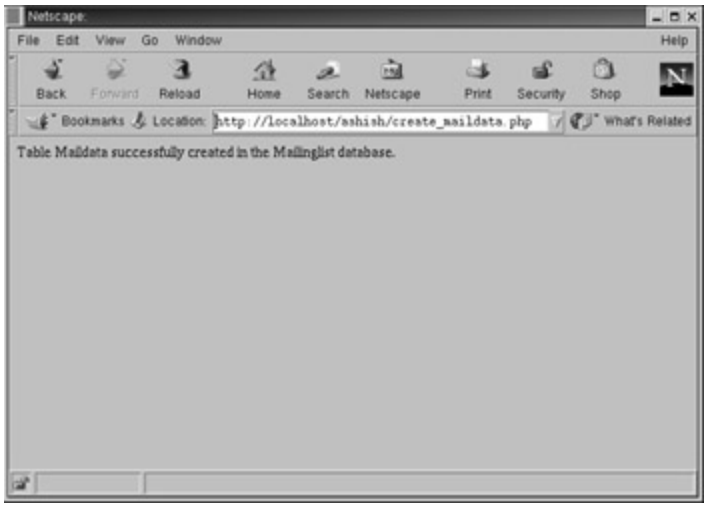

Figure 13-4: Creating a table in the database

Figure 13-5 exhibits the error message that is displayed if you are unable to select the database where you want to c the table.

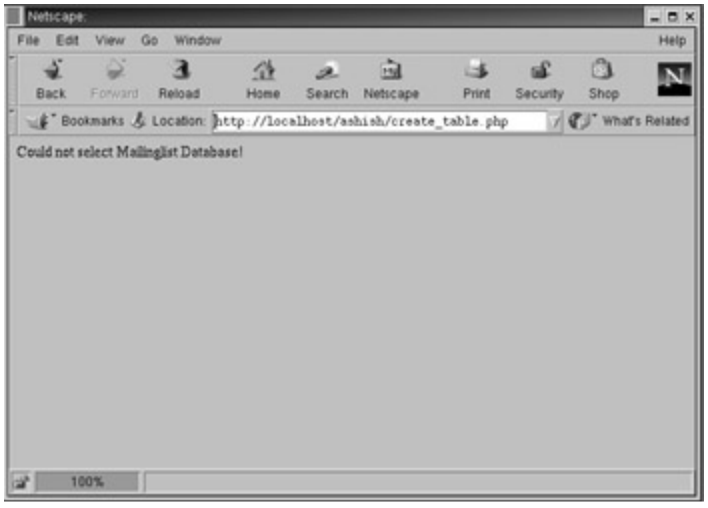

Figure 13-5: Error displayed when creating a table in the database

You now know how to create a table in MySQL. In the next section , you will learn how to insert records in the table.

Inserting Records in a Table

So far, you have learned how to connect to MySQL and gain access to the database. You have also learned to creat database and a table in the database. Now you will learn how to insert information into the table. The information tha be stored in the databases is accepted through the HTML Web page. You must save this file in the /var/www/html directory. Following is the code that you will use to create the HTML file to accept the information. You will name the file as `add_row.html` .

```
                    <html>
                   <head>
                   <title>Newsmail registration form</title>
                   <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
                   </head>

                   <body bgcolor="#FFFFFF" text="#000000">
                   <p> </p>
                   <form name="Mailinglist" method="post" action="addrow.php">
                   <table width="287" border="0" align="center" cellpadding="3">
                   <tr bgcolor="#FFFFFF">
                   <td colspan="2">
                   <div align="center"><b>Enter mailing list information
</b></div>
                   </td>
                   </tr>
                   <tr bgcolor="#FFFFFF">
                   <td width="107">
                   <div align="right">Name </div>
                   </td>
                   <td width="162">
                                       <input type="text" name="name" size="25">
                   </td>
                   </tr>
                   <tr bgcolor="#FFFFFF">
                   <td width="107">
                   <div align="right">e-mail</div>
                   </td>
                   <td width="162">
                                      <input type="text" name="E-mail" size="25">
                   </td>
                   </tr>
                   <tr bgcolor="#FFFFFF">
                   <td width="107">
                   <div align="right">Secondary e-mail</div>
                   </td>
                   <td width="162">
                                      <input type="text" name="Secondaryemail"
size="25">
```

```
                    </td>
                    </tr>
                    <tr bgcolor="#FFFFFF">
                    <td colspan="2">
                    <div align="center">
                                        <input type="submit" name="Submit"
value="Submit">
                    </div>
                    </td>
                    </tr>
                    </table>
                    </form>
                    </body>
                    </html>
```

In the code given above, the created Web page accepts the name, the e-mail ID, and the alternative e-mail ID of a u After the user has finished entering the required information, the user clicks on the Submit button. When the button i clicked on, all the entered information is transmitted to the MySQL server. As you may notice in the code, the informa transmitted to the PHP script by using the POST method. The Web page appears as shown in Figure 13-6 .

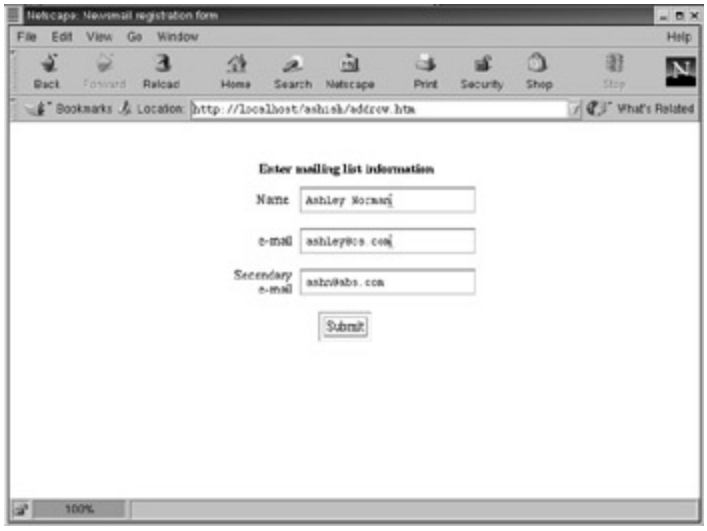

Figure 13-6: Accepting information from users

Now the information is sent to the MySQL server for inserting the information in the Mailinglist table in the form of rec The following steps have to be performed to insert the information into the table. You will

- Use the `mysql_connect()` command to establish a connection with MySQL.
- Use the `mysql_select()` function to select the `Mailinglist` database.
- Use the `mysql_query()` function to send the information to the MySQL server database. The information is pa in the form of a `select` query that is stored in a variable in the form of a string. This variable is passed as a parameter in the `mysql_query()` function. The variable will contain the values True or False based on whethe information is successfully inserted into the table. If the variable contains the value False, an appropriate error message is displayed. You will now create the actual PHP script that will handle the task of inserting the informa the table. The script will be saved as addrow.php.

```
                    <html>
```

```
<head>
<title>Add a new row in  mail data table</title>
</head>
 <body bgcolor="#FFFFFF" text="#000000">
<p> </p>

<?php

                                    $connect=
mysql_connect("localhost","root","")
                                       or die("Could not connect to database
in localhost !");

                                    $result=mysql_select_db("Mailinglist")
                                       or die("Could not select Mailinglist
database !");

                                    $sqlquery = "INSERT INTO Maildata VAL
UES(‘". $Name ."','". $Email . "‘,'".
                                    $Secondaryemail."')";

                                    $queryresult = mysql_query($sqlquery) or
die(" Could not execute mysql
                                    query !");

                                    echo "<table border=1 align=center  width=500>"
                                    echo "  <tr> ";
                                    echo "    <td> ";
                                    echo "     Name";
                                    echo "    </td>";
                                    echo "    <td>".$name. "</td>";
                                    echo "  </tr>";
                                    echo "  <tr> ";
                                    echo "    <td > ";
                                    echo "      Email";
                                    echo "    </td>";
                                     echo "    <td >".$Email. "</td>";
                                    echo "  </tr>";
                                    echo "  <tr> ";
                                    echo "    <td > ";
                                    echo "      Secondary e-mail";
                                    echo "    </td>";
                                    echo "    <td>".$Secondaryemail. "</td>";
                                    echo "  </tr>";
                                    echo "</table>";


?>
</body>
</html>
```

In the code given above, a connection is established with MySQL by using the `mysql_connect()` function. Next yo

access to the database Mailinglist. The database contains a table named Maildata. You will insert the record in this ta
You created the select query and stored it in the variable `$sqlquery` . The string contains the name, e-mail address
secondary e-mail address of the user. This information will be used later in the book to create a mailing list and send
information to a group of registered users. In the `select` statement, you may observe that each piece of information
linked with the specific field in the table where the information should be saved. You must specify the information in th
same sequence in which the fields are created in the table. If the `select` query is executed successfully and the
information is inserted into the table, the variable `$queryresult` will contain the value True. If the variable contains
value True, then the output is displayed as shown in Figure 13-7 . However, if the variable contains the value False, a
error message is displayed and the user exits the Web page. The variable will contain the value False in the situation
given below.

Figure 13-7: Displaying accepted information

- The e-mail address already exists.
- Another user has exclusively locked the table.
- The MySQL server may be overloaded and may be unable to handle any more queries.
- The user does not have the required write permission to insert information in the table.

After inserting information into the database, you may want to view specific information from a table. The next section
covers this topic.

Retrieving Information from a Table

Now that you have established a connection with the database and stored some information, you may want to retriev
information from the database. However, you cannot directly extract information from the database. To get the neces
information, you need to create queries written in *Structured Query Language* (SQL). These queries extract the neces
information from the database and return the information as records.

MySQL provides the `mysql_query()` function to transmit the query to the server and return the result to the calling
program. The function takes the query as an argument and passes it to the MySQL server. The server then processe
query and returns the requested information. The syntax of the `mysql_query()` function is given below.

```
$queryres = mysql_query($query_string);
```

In the preceding code, the `mysql_query()` function takes the variable `$query_ string` as a parameter and store result in `$queryres` . The variable `$query_string` should contain the formatted query written in SQL. The variabl `$queryres` contains the value True if the select query is executed successfully. The variable can contain the value only in the following conditions:

- Users may not have the necessary permission to access the queried information.
- There may be a syntax error in the select query.
- The queried information may not be available.

Note You do not need to add a semicolon to the select query while assigning it to the variable.

You need to perform the following steps to extract information from a table in MySQL.

1. You must first connect to MySQL by using the `mysql_connect()` function.
2. Next you have to select the database in which the information exists by using the `mysql_select_db()` func
3. Next you need to create a select query to insert or retrieve the necessary information from a table in the datab In Chapter 11 , 'Handling Files ,' you learned about creating queries by using the `select` statement. The strir passed as a parameter to the `mysql_query()` function, which then sends the query to the MySQL server.
4. Finally the query is executed on the MySQL sever and a value is returned and stored in a variable. The variab contains the value True if the select query has been successfully executed, and the value False if it has not executed successfully. In case of failure, an error message is displayed.

The code given next contains a select query that retrieves all the records stored in Maildata. The output of the code i shown in Figure 13-8 .

```
 <html>
<head>
<title> Displaying Records Retrieved From A Table </title>
</head>
<?php

 $connect= mysql_connect("localhost","root","")
or die("Could not connect to MySQL server
in localhost !");

$result=mysql_select_db("Mailinglist")
   or die("Could not select Mailinglist Databas

echo "Database Mailinglist was
successfully selected.", "\n";
echo "<BR>";
$sqlquery = "SELECT * from Maildata";
$queryresult = mysql_query($sqlquery)or die
("Query Failed");

 echo "<BR>";
 echo "All the records from the table Maildata
retrieved successfully.", "\n";

?>
```

```
</html>
```

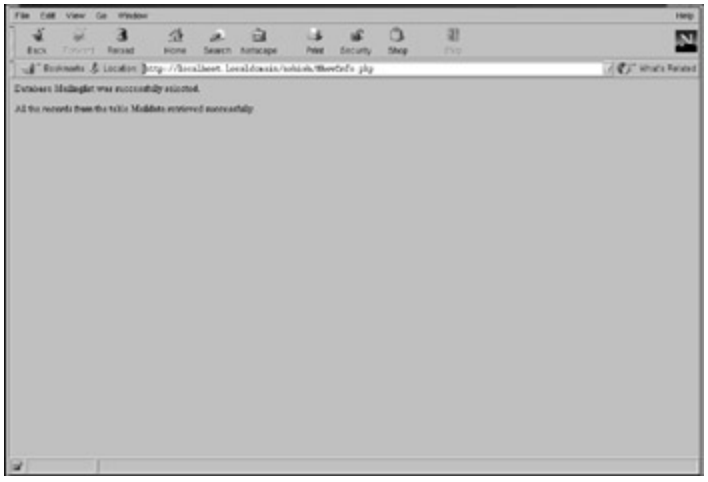

Figure 13-8: Querying in a database in MySQL

The code connects to MySQL by using the `mysql_connect()` function and accesses the Mytestdb database by us the `mysql_select_db()` function. After the connection has been established, an SQL query is created, which is st in the `$query_string` variable. This string is then passed to the MySQL server by using the `mysql_query()` func If the query is executed successfully, the variable `$query_result` contains the value True.

PHP provides many functions to retrieve the information if the query executes successfully. These functions process row of information separately. Therefore, you need to call the function as many times as the number of rows in the re For example, if the query returns 20 rows of data, you will call the function 20 times. Instead of writing the statement times, you can use the `mysql_affected_rows()` function to retrieve the number of affected rows and store the information in a variable. You can then use the variable to keep track of the number of rows returned and the numbe rows processed. You will learn more about this function later in the chapter. You can then use a `while` loop, which w until all the records in the result are traversed and displayed on the Web page. The following two functions are used PHP to retrieve the information from the result.

- **The mysql_fetch_row() function.** This function returns the result as an enumerated array. While using an enumerated array, the information is fetched from the result in the form of an indexed array. The array stores the field from the result at `index1` , the next field at `index2` , and so on.

- **The mysql_fetch_array() function.** This function returns the result as an associative array. In the case of assoc arrays, the fields are indexed based on their field names. If an error is encountered and no value is retrieved, the function returns the value NULL.

Now consider these functions in detail.

The *mysql_fetch_row()* function

The `mysql_fetch_row()` function processes the information from the result one record at a time. The information stored in the format of an enumerated array and each field of the record is stored as an element of the array. The syn the function is given below.

```
$Resrow = mysql_fetch_row($result);
```

As explained earlier, the `$Resrow` variable indicates whether the select query has executed successfully. The variab `$Resrow` stores the values in the same order in which the information is received. Therefore, the value stored in the

field of the result is assigned to the first element of the array $Resrow and the second field is stored in the second element. Similarly, all the other values retrieved in the result are assigned to their respective fields. The number of ro the array is the same as the number of rows in the result.

When you use the mysql_fetch_row() function to retrieve information, the following steps are performed:

1. You will connect to the database by using the mysql_connect() function.
2. Next you will select the database named Mailinglist by using the mysql_select_db() function.
3. Next you create the SQL query to retrieve the information from the database. This query is stored in the variab $MyQuery .
4. When this query is executed, all the records from the Mailinglist table in the Mailinglist database are retrieved.
5. The $Result variable contains the value that indicates whether the query has been executed successfully. T variable contains the value True to indicate that the execution has been successful. The variable will contain t value False in any of the following circumstances:
   - A user gains exclusive access to the table by using an exclusive lock.
   - The MySQL server is unable to execute any more SQL queries.
   - The table used in the query statement does not exist.
   - The user who is requesting the information does not have the required read permission.
6. Create a variable named $Resrow and store the number of records returned in the result by executing the mysql_effect_rows() function.
7. Use a while loop to execute the loop until $Resrow does not return zero.
8. The variable $Resrow is a one-dimensional array and its first element stores the value of the first field of the Mailinglist table.
9. The while loop retrieves each row from the Mailinglist table and displays the information on the Web pa

In the next section , you will learn to perform the same action by using the mysql_fetch_array() function.

The *mysql_fetch_array()* function

In the previous section , you learned to retrieve the information from the result by using the mysql_fetch_row() function. In this section, you will use the mysql_fetch_array() function to retrieve information from the resultset record at a time. However, in the case of the mysql_fetch_array() function, the records are retrieved in the form associated array. In the case of an associated array, each element of the array contains a field from a row of the resu Each element of the array is named by its corresponding field name in the resultset. The syntax of the mysql_fetch_array() function is given below.

```
$Resrow = mysql_fetch_array($result);
```

In the syntax given above,

- The $result , which is also known as the result identifier, contains the resultset returned by the mysql_query function.
- The $Resrow variable is an associated array that will contain a row from the result returned by the mysql_fetch_array() function.

- The first element of the `$Resrow` variable contains the value in the first field of the resultset. The label of the ele is the same as the field name. Similarly, all the other elements of the variable also contain their corresponding fie values and names.

The advantage that the `mysql_fetch_array()` function has over the `mysql_fetch_row()` function is that in the of the `mysql_fetch_array()` function, you do not need to remember the actual order in which the fields are place the table.

You need to perform the following steps if you use the `mysql_fetch_array()` function to retrieve information:

1. A connection is established with MySQL by using the `mysql_connect()` function.
2. Next you connect to the `Mailinglist` database by using the `mysql_select_db()` function.
3. Then the select statement is created that will extract the required information from the table `Mailinglist,` v is in the `Mailinglist` database.
4. The query is stored in the `$MySQLQuery` variable.
5. The `$MySQLQuery` variable is passed as a parameter to the `mysql_query()` function, which executes the q and returns and stores the result in the `$result` variable.
6. If the query is successful, the variable `$result` contains the value True. The variable can contain the value F for any of the following reasons:
    - Another user has locked the table.
    - The table does not exist.
    - The MySQL server is unable to handle any more select queries.
    - The user doesn't have the necessary read permission for the accessed table.
7. Create a variable named `$Resrow` and store the number of records returned in the result by executing the `mysql_effect_rows()` function.
8. Use a `while` loop to execute the loop until `$Resrow` does not return zero.
9. The variable `$Resrow` is a one-dimensional array and its first element stores the value of first field of the Mail table. The field name is assigned as the label for the element.
10. The `while` loop retrieves all the information from the Mailinglist table by using the `mysql_fetch_array()` function and displays it on the Web page.

You will now learn how you can update information that already exists in a database.

Updating Information in a Table

You have learned how to insert information into a table. In the same way, you can also update information stored in a table. You use the `update` command to perform updating in a table in MySQL. While updating information in a table good idea to first display the existing information before changing it. The following steps are performed while updatin information in a table.

1. You must first accept the information in an HTML page based on which information needs to be updated in the table. This information should be from a field in the table that has unique records, which is mostly the primary
2. When the user clicks on Submit, a PHP script is executed.

3. This script establishes a connection with the MySQL database server and selects the database by using the `mysql_select_db()` function.
4. The query is then sent to the MySQL database server by using the `mysql_query()` function. The query is fu qualified by using the `where` clause. The condition used in the `where` clause ensures that only a single recor stored in the result identifier.
5. The information is then stored in the result identifier and returned to the browser.
6. The information is then displayed in the HTML form created by the PHP script.
7. The form control on the HTML page displays the values from the returned result. The value is stored in a field to one form control. The form is displayed in the browser.
8. The user makes the necessary changes in the information and clicks on the Submit button to indicate that the information should now be updated.
9. Another PHP script is now called that establishes a new connection with the database and updates the inform in the required table by using the update command in the select query. The query is executed by using the `mysql_query` function.
10. The result identifier indicates whether the information has been successfully updated. If the information is successfully updated, then the successfully updated message appears, otherwise an error message is displa

The code given below creates an HTML form that displays all the records available in the Maildata table. Each record contain an Edit hyperlink. The user clicks on the hyperlink to make changes in the record. When the hyperlink is click on, the value stored in the `Email` field of corresponding records is passed to the next HTML page. The new HTML p will display the record for editing. Save this code in a file named `db_browser3.php` .

```
<html>
<head>
<title>Maildata record browser</title>
</head>

<body bgcolor="#FFFFFF" text="#000000">

<?php

                                    $connect=
mysql_connect("localhost","root","")
                                    or die("Could not connect to MySQL server
in localhost !");

                                    $selectdb=mysql_select_db("Mailinglist")
                                    or die("Could not select mail data data
base !");

                                    $sqlquery = "SELECT * from Maildata";

                                    $queryresult = mysql_query($sqlquery);

                                    echo "<table width=700 border=1 align=cen
ter>";
                                    echo " <tr>";
```

```
                                        echo "  <td width=200> <center><b>Name
</b></center></td>\n";
                                        echo "  <td width=200> <center><b>e-mail
</b></center></td>\n";
                                        echo "  <td width=200>
<center><b>Secondary e-
                                            mail</b></center></td>\n";
                                        echo "  <td width=100> <center><b>Action
</b></center></td>\n";
                                        echo "  </tr>\n";
                                        while ($row=mysql_fetch_array
($queryresult))
                                        {
                                                        echo "  <tr>\n";
                                                        echo "    <td>".$row
["Name"]."</td>\n";
                                                        echo "    <td>".$row
["Email"]."</td>\n";
                                                        echo "    <td>".$row
["Secondaryemail"]."</td>\n";
                                                        echo "    <td><A
                                                href=\"displayform.php?Email=".
$row["Email"]."\">edit</a></td>\n";
                                                        echo "  </tr>\n";
                                        }
                                        echo "</table>\n";
                    ?>
                    </body>
                    </html>
```

The output of the code given above appears in Figure 13-9 .

Figure 13-9: Displaying all the available records for editing

The Web page created by using the above will display the record's information for editing. Once the user has made tl
necessary changes, they click on the Submit button. When the Submit button is clicked on, this information is sent to

MySQL server where the information is updated in the database. You will save the code in a file named display.php. output of the code given below is shown in Figure 13-10 .

```
<html>
<head>
<title>Mail record update form</title>
</head>

<body bgcolor="#FFFFFF" text="#000000">
 <form name="maildata" method="post" action="update.php">
<table width="250" border="1" align="center">

<?php

$connect=
mysql_connect("localhost","root","")
or die("Could not connect to MySQL server
in localhost !");


$result=mysql_select_db("Mailinglist")
or die("Could not select Mailinglist
Database");


$sqlquery = "SELECT * from Maildata
where Email='" .$Email ."'";


$queryresult = mysql_query($sqlquery);

if($row=mysql_fetch_array($queryresult))
   {
                    echo "<tr> ";
                    echo " <td> Name </td>";
                    echo " <td width=\"150\">
 ". $row["Name"] . " </td>";
                    echo "</tr>";
                    echo "<tr> ";
                    echo " <td> e-mail </td>";
                    echo " <td>". $row["Email"]
."</td>";
                    echo " <td> <input type=\
"hidden\" Name=\"Email\"
                              value=\"".$row
["Secondaryemail"]."\" ></td>";
                    echo "</tr>";
                    echo " <tr> ";
                    echo " <td> secondary
e-mail </td>";
                    echo " <td> ";
                    echo "  <input type=\"text\"
Name=\"Secondaryemail\"
                    value=\"".$row["Secondaryema
"\" >";
                    echo " </td>";
```

```
                                                    echo "</tr>";
                                                    echo "<tr> ";
                                                    echo " <td> ";
                                                    echo "  <center> ";
                                                    echo "  <input type=\"submit\"
name=\"Submit\"
                                                    value=\"Submit\">";
                                                    echo "  </center>";
                                                    echo " </td>";
                                                    echo "</tr>";
                                            }
                    ?>
                    </table>
                     </form>
                    </body>
                    </html>
```

Figure 13-10: Displays the record selected for modification

The updated information is then retrieved from the MySQL database and displayed on a new HTML page. You will sa
this code in a file named changes.php. The code is given below, and the output of the final HTML page appears in Fi
13-11 .

```
                    <html>
                    <head>
                    <title>New record In Newsmail table</title>
                    <meta http-equiv="Content-Type" content="text/html;
charset=iso-8859-1">
                    </head>
                    <body bgcolor="#FFFFFF" text="#000000">
                    <p> </p>

                    <?php

                                        $connect= mysql_connect("localhost","root","")
                                                  or die("Could not connect to
```

```
Database !");

                                        $result=mysql_select_db("Mailinglist")
                                                    or die("Could not select
Maildata database");


                                        $sqlquery = "UPDATE Maildata SET
Secondaryemail=\"".
                                         $secondaryemail."\" where Email=\"".
$Email."\"";


                                        echo $sqlquery;
                                        $result = mysql_query($sqlquery)
                                        or die("Could not execute SQL query");

                                        echo " Record was successfully updated.";

                    ?>
                     </body>
                    </html>
```

Figure 13-11: Updating the changed user information

In the code given earlier, you learned how to connect to a single database and how to display information stored in a that exists in the database. Suppose you need to display information available in multiple databases. PHP provides a function to establish connections with multiple databases. You will now consider this procedure in detail.