PHP (Personal Home Page) – A Basic Introduction

Ever since Web designers found out about the tag, the Internet has seen an explosion in the number of Web sites that depend heavily on user response and interactivity. For a long time, the primary language used to develop such Web sites was Perl. But ask any novice programmer, and he'll tell you that learning Perl isn't exactly fit and easy to tackle complicate web development.

As a result, there has been a proliferation of alternative server-side scripting languages, which perform many of the tasks previously handled by Perl, but have a shorter learning curve. The most well-known of these are ASP and PHP; while the former works primarily on the Windows platform in combination with a clutch of proprietary products, the latter has the unique distinction of being an open-source server-side scripting language that's both fun and easy to learn. Today, it is estimated that more than 1,000,000 Web sites use PHP as a server side scripting language.

PHP was first developed by **Rasmus Lerdorf** as a means of monitoring page views for his online resumé, and slowly started making a mark when PHP/FI was released in mid-1995. This version of PHP had support for some basic Web functions - the ability to handle form data, support for the mSQL database, and more.

As PHP's popularity grew, the development of the language shifted from Rasmus to a team of dedicated programmers who took upon themselves the onus of rewriting the PHP parser from scratch. The result of the efforts was PHP 3.0, which included support for a wider range of databases, including MySQL and Oracle. And PHP 4.0, which was released a few weeks ago, uses the powerful new Zend scripting engine to deliver better performance, supports Web servers other than Apache, and comes with in-built support for session management.

My goal in this chapter is very simple - I'll introduce to you the basics of using PHP to power your Web site, and related Web database development efforts. The only assumptions we're going to make throughout this series are that you know the basics of HTML, are using a properly configured Web server running PHP4.You can download the latest distribution of PHP4 from the official PHP Web site at http://www.php.net/ and take a look at the installation instructions. Finally, because of limited time, those advanced PHP programming techniques such as array manipulation, programming with PHP class, function, etc…is not going to discuss in this chapter.

7.1 Basic Syntax

There are two main differences between a standard HTML document and a PHP document. First, PHP scripts should be saved with the ".php" extension. Second, you wrap your PHP code with the "<?PHP" and "?>" tags to indicate what is PHP as opposed to what is HTML.

```
<HTML>
<HEAD>
<TITLE>First PHP Script</TITLE></HEAD>
<BODY>
<?php
print("<h1>PHP test</h1><p>"); //sending HTML tags
?>
<?
phpinfo(); //calling PHP function
?>
</BODY></HTML>
```

Script1. This is the most basic structure of an HTML document, with the PHP tags inserted into the body section. All PHP scripts must use some form of the PHP tags in order for the server to know what to process as PHP, while everything outside of them gets sent to the browser as standard HTML.

7.2 Variables

A variable is best thought of as a container for data. Once data has been stored in a variable (or, put differently, once a variable has been assigned a value), that data/variable can be altered, printed to the Web browser (when I say printed, it may help to think of it as sent, but it's the print statement that does the sending, so either term is appropriate), saved to a database, e-mailed, and so forth.

Variables are, by their nature, flexible: you can put data into a variable, retrieve that data from it (without affecting the value of the variable itself), put new data in, and you can continue this cycle as long as is necessary. But, variables in PHP are also temporary: they only exist - that is, they only have a value - while they are used within a script. Once you are in a new page, those variables cease to exist, unless you pass them along to the new page, which I'll discuss in the next chapter (HTML Forms and PHP).

In PHP all variables begin with a dollar sign ($), followed by the variable name itself. This name must begin with either a letter (A-Z, a-z) or the underscore (_), followed by any number of letters, underscores, or numbers, used in combination or not. You may not use spaces within the name of a variable. Instead, the underscore is commonly used to separate words in a variable name.

Keep in mind that variables are case-sensitive. Consequently, "$variable" and "$Variable" are two different constructs, although it would never make sense to use two variables with such similar names. One should quickly get into the habit of creating variable names that make sense on their own, as well as using comments to indicate the purpose of variables. These habits will reduce errors and make revisiting your work less taxing. For example, "$FirstName" is more useful than "$FN" and putting in a comment that details what a variable's purpose is will make your work abundantly clear. In fact, you may decide that "$first_name" is a better variable name than "$FirstName" because there are no capital letters to get right and the words are separated for clarity. No matter how you decide to name your variables, the most important thing to remember is that whatever convention you use, be consistent. This will help you avoid making trivial errors in your programming.

Unlike some other languages, in PHP you neither have to declare what a variable is (to declare a variable is to assign it a type - I'll cover variable types in Types of Variables) nor initialize it prior to first use (to initialize a variable is to create it). With PHP a variable exists and is defined the first time you use it.

Numbers

I've combined the two types of numbers (integers and floating-point) into one group for ease of learning. I'll discuss the difference between the two briefly.

The first type of numbers (integers) is the same thing as whole numbers. They can be positive or negative but include neither fractions nor decimals. Numbers which use a decimal point (even such as "1.0") are floating point numbers. You must also use floating point numbers to refer to fractions, since the only way to express a fraction within PHP is to convert it to its decimal equivalent. So "1 1/4" would be written as "1.25".

Examples of valid integer values include:
1
1972
-1

Examples of valid floating-point values include:
1.0
19.72
-1.0

Since we will refer to both as numbers here, any of the above would be considered valid number values. Examples of invalid number values would include:
1 1/4
1972a
02.23.72

The fraction is invalid as it contains two unusable characters: the space and the slash (/).The second item is invalid as it uses both numbers and letters, which is acceptable for the name of a variable, but not as the value of a number variable. The third example is invalid since it uses two decimal points. If you need to refer to one of these values for some reason other than to perform calculations on them, you can assign them as strings.

Strings

A variable is a string if it consists of characters (some combination of letters, numbers, symbols, and spaces) enclosed within either a pair of single (') or double (") quotation marks. Strings can contain any combination of characters, including other variable names.

Examples of valid strings values include:
"Hello, world!"
"Hello, $FirstName!"
"1 1/4"
'Hello, world! How are you today?'
"02.23.72"
"1972"

Notice how in the last example you took an integer and made it into a string by putting it within quotes. Essentially the string contains the characters "1972" whereas the number is equal to 1972. It's a fine distinction and one that will not matter in your code, as you could perform mathematical calculations with the string "1972" just as you could with the number.

Examples of invalid string values include:
Hello, world!
"I said, "How are you?""

The first example is invalid as it is not within either single or double quotes. The second example is tricky. You will have problems assigning that value to a string because once PHP reads the second quotation mark, it assumes that the string ends there and the continuing text will cause an error.

Then how do you use a quotation mark within a string you may wonder? Just as discussed in previous section when using the print() function to create HTML, you can escape the quotation mark by putting a backslash (\) before it. By changing this string to "I said, \"How are you?\"", you have told PHP to include those two quotation marks as part of the value of the string, and not treat them as the string opening or closing indicators. So while any combination of characters can be included in a string, special characters must be escaped to print correctly. Along with the double quotation mark, you should also escape the apostrophe or single quotation mark (`), the backslash(\), and the dollar sign ($).

Array

Arrays constitute a complicated but very useful notion: they are a collection of multiple values assembled into one overriding variable. An array can consist of numbers and/or strings (and/or other arrays), which allows this one variable to hold exponentially more information than a simple string or number ever could. For example, if you wanted to create a grocery list using strings, your code would look something like:
$Item1 = "apples";
$Item2 = "bananas";
$Item3 = "oranges";

For each added item, you would need to create a new string. This is cumbersome and it makes it difficult to refer back to the entire list or any specific value later in your code. You can greatly simplify matters by placing your entire list into one array (say, $Items), which contains everything you need to put on that list. As an array, your list can be augmented, sorted, searched, and so forth.

7.2 HTML Forms and PHP

Perhaps the most common use of variables is in conjunction with HTML forms. Web sites utilize forms to register and login users, to receive feedback, for online shopping, and for many other purposes. Even the most basic site will find logical reasons to incorporate them.

Frequently, programmers create CGI scripts in Perl to handle the data created by these forms, but the same results can be achieved more easily using PHI? Unlike CGI scripts, where you have to write a segment of code that will extract the information sent by the form, PHP has a nice method of built-in support for receiving data from an HTML form without the need of any parsing.

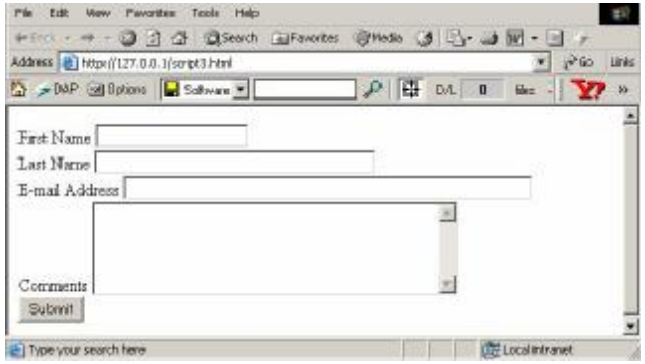
This section covers the basics of creating HTML forms and how that data is transmitted to your PHP script. Those who are new to the topic of forms may want to refer to an in-depth HTML resource for more detailed coverage of the topic, considering their importance in the field of Web site design.

For your HTML form you will create a feedback page that takes the user's first and last names, e-mail address, and comments. You'll need to create the necessary fields with this in mind.

To create an HTML form:

```
<HTML><HEAD><TITLE>HTML Form</TITLE></HEAD>
<BODY>
<FORM ACTION="script2.php" METHOD=POST>
First Name <INPUT TYPE=TEXT NAME="FirstName" SIZE=20><BR>
Last Name <INPUT TYPE=TEXT NAME="LastName" SIZE=40><BR>
E-mail Address <INPUT TYPE=TEXT NAME="Email" SIZE=60><BR>
Comments <TEXTAREA NAME="Comments" ROWS=5 COLS=40></TEXTAREA><BR>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

Script2. Any combination of input types can be added to your form-just ensure that all of them are within the <FORM> tags or else those elements will not appear. As a stylistic suggestion, laying out these input elements within a table can give your form a more professional and useable appearance.



If you have typed in your form correctly, it should look like this in your web browser. Make sure that you close the form and include the submit button within it.

The experienced reader will notice that we are missing one thing in our initial <FORM> tag, namely adding a METHOD attribute. This attribute tells the server how to transmit the data from the form to the handling script. I omitted it earlier because the topic merits its own discussion.

There are two choices you have with METHOD: GET or POST. I suspect that most HTML coders are not entirely clear on the distinction and when to use which. In truth, for the most part it won't make much difference (especially as you first begin using them) as either will generally get you the results you need.

The difference between using GET versus POST is squarely in how the information is passed from the form to the processing script. The GET method will send all the gathered information along as part of the URL. The POST method transmits the information invisibly to the user. For example, upon submitting your form, if you use the GET method, the resulting URL will be something like:

```
http://www.DMCinsights.com/php/script3.php?FirstName=Larry&LastName=Ullman
```

Whereas using the POST method, the end user would only see

```
http://www.DMCinsights.com/php/script3.php
```

When choosing which method to use, you may want to keep in mind these three factors:

1. With the GET method you are limited as to how much information can be passed;
2. The GET method publicly sends the input to the handling script (which means that, for example, a password which is entered in a form becomes viewable by anyone within eyesight of the Web browser, creating a larger security risk); and,
3. A page generated by a form that used the GET method can be bookmarked while one based upon POST cannot be.

To receive and process the data, we need the following PHP script.

```
<HTML>
<HEAD><TITLE>Form Results</TITLE>
<BODY>
<?php
/* This page receives and handles the data generated by "script2.html". */
print "Your first name is $FirstName.<BR>\n";
print "Your last name is $LastName.<BR>\n";
print "Your E-mail address is $Email.<BR>\n";
print "This is what you had to say:<BR>\n $Comments<BR>\n";
?>
</BODY>
</HTML>
```

Script2. Print out the content captured in the HTML form.

7.3 Using Numbers

Just as you learned in grade school, the most basic mathematics involved the principles of addition, subtraction, multiplication, and division. To demonstrate these principles, you'll create a PHP script that calculates the total cost for the sale of some widgets. This script could be the basis of a shopping cart application-a very practical Web page feature.

```
<HTML ><BODY>
<FORM ACTION="script3.php" METHOD=POST>
Cost <INPUT TYPE=TEXT NAME="Cost" SIZE=10><BR>
Quantity <INPUT TYPE=TEXT NAME="Quantity" SIZE=10><BR>
Discount <INPUT TYPE=TEXT NAME="Discount" SIZE=10><BR>
Tax <INPUT TYPE=TEXT NAME="Tax" SIZE=10><BR>
Installment <INPUT TYPE=TEXT NAME="Installment" SIZE=10><BR>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit">
</FORM></BODY></HTML>
```

Script3. HTML input for Cost, Quantity, Tax, Discount and Installment.


```
<HTML><HEAD><TITLE>Using Numbers</TITLE>
</HEAD><BODY>
<?php
//Define the constants
$Cost=abs($Cost);//Enusure positive input
$Quantity = abs($Quantity);//Ensure positive input
$Discount = abs($Discount);
$Tax = abs($Tax);
$Installment = abs($Installment);
$Installment = round($Installment,0);
//Calculation
$TotalDiscount = $Cost*$Quantity*$Discount;
$TotalCost = $Cost * $Quantity;
$Tax = $Tax++; // $Tax is now worth 1.06.
$TotalCost = $TotalCost - $TotalDiscount;
$TotalCost = $TotalCost * $Tax;
$Payments = $TotalCost / $Installment;
$Tax = $Tax--;
// Print the results
print ("You requested to purchase $Quantity INTEL P4 1.7 GHz CPU(s) at \$$Cost each.\n<P>");
print ("For you, we offer $Discount discount for cost and the discount is \$$TotalDiscount.\n<p>");
print ("The total with $Tax tax, minus your \$$TotalDiscount, comes to \$$TotalCost. \n<P>");
// Print with format. 2 significant figures or 1 trailing zero
print ("You may purchase the CPU(s) in $Installment monthly installments of \$");
printf ("%01.2f",$Payments);
print (" each.\n<P>");
?>
</BODY></HTML>
```

Script3. While the calculations themselves are straightforward, you should feel free to add any other comments you feel necessary to illuminate the process here.

7.4 Using String

Trimming Strings

Either because a user entered information carelessly or because of sloppy HTML code, it's quite common for extra spaces to be added to a string variable. For purposes of clarity, data integrity, and Web design, it is worth your while to delete those spaces from the strings before using them. Extra spaces sent to the Web browser could make the page appear oddly and those sent to a database or cookie could have unfortunate consequences at a later date (e.g., if a password has a superfluous space it might not match when entered without the space).

The trim() function automatically strips away any extra spaces from both the front and the end of a string (but not within the middle). The format for using trim() is:

```
$String = " extra space before and after text ";
$String = trim($String);
// $String is now equal to "extra space before and after text"
```

If you need to trim off excess spaces from the beginning or the end of a string, but not both, PHP has broken the trim() function down into two more specific ones: rtrim() will remove those spaces found at the end of a string variable and ltrim() will handle those at the beginning.

```
<HTML><HEAD><TITLE>HTML Form</TITLE></HEAD>
<BODY>
<FORM ACTION="script4.php" METHOD=POST>
First Name <INPUT TYPE=TEXT NAME="FirstName" SIZE=20><BR>
Last Name <INPUT TYPE=TEXT NAME="LastName" SIZE=40><BR>
E-mail Address <INPUT TYPE=TEXT NAME="Email" SIZE=60><BR>
Comments <TEXTAREA NAME="Comments" ROWS=5 COLS=40></TEXTAREA><BR>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

Script4. HTML for capturing the inputs

```
<HTML><HEAD><TITLE>Form Results/Using Strings</TITLE>
<BODY>
<?php
/* This page receives and handles the data generated by "script4.html". */
$FirstName = trim($FirstName);
$LastName = trim($LastName);
$Email = trim($Email);
$Comments = trim($Comments);
print ("Your first name is $FirstName.<BR>\n");
print ("Your last name is $LastName.<BR>\n");
print ("Your E-mail address is $Email.<BR>\n");
print ("This is what you had to say: <BR>\n $Comments<BR>\n");
 ?>
</BODY></HTML>
```

Script4. Using TRIM function to remove the extraneous spaces.

Connecting Strings (Concatenation)

It's an unwieldy term, but a useful concept, **concatenation**. It refers to the process of linking items together. Specifically in programming, you concatenate strings. The period (.) is the operator for performing this action, and it's used like so:

$NewString = $aString . $bString.

You can link as many strings as you want in this way. You can even join numbers to strings:
$NewString = $aString . $bString . $cNumber;

This works because PHP is weakly typed, meaning that its variables are not locked in to one particular format. Here, the $cNumber variable will be turned into a string and appended to the value of the $NewString variable.

```
<HTML><HEAD><TITLE>Form Results/Using Strings</TITLE><BODY>
<?php
/* This page receives and handles the data generated by "script5.html". */
$FirstName = trim($FirstName);
$LastName = trim($LastName);
$Email = trim($Email);
$Comments = trim($Comments);
$Name = $FirstName . " " . $LastName;
print ("Your name is $Name.<BR>\n");
print ("Your E-mail address is $Email.<BR>\n");
print ("This is what you had to say: <BR>\n $Comments<BR>\n");
?></BODY></HTML>
```

Script5. Use the Script4.html and apply concatenation to “join” the string.

7.5 Control Structures

The If Conditional

The basic programming conditional is the standard if (what used to be called an if-then conditional-the then is now implied).

The syntax for this kind of conditional is very simple:

```
if      (condition) {
        statement(s);
}
```

The condition must go within parentheses and then you begin the statements area after a curly brace. The statements section of the conditional is where executable commands are placed (for example, printing a string, adding two numbers together, and so forth). Each separate statement (or command) must have its own semicolon indicating the end of the command line, but there is no limit to how many statements you write as the result of a conditional. You then close the statements section with another curly brace. Commonly programmers put these statements indented from the initial if line to indicate that they are the result of a conditional but that is not syntactically required. Failure to use a semicolon after each statement, forgetting an opening or closing parentheses or curly brace, or using a semicolon after either of the braces will all cause errors to occur. PHP uses the concepts of TRUE/FALSE when determining whether or not to execute the statements. If the condition is TRUE, the statements will be executed; if FALSE, they will not be. The next section, More Operators, goes into TRUE/FALSE in more detail.

Using Else

The next logical formation after an if conditional is the if-else (sometimes called the if-then-else) conditional. This allows you to establish a condition as to why one statement would be executed and then another statement which would be executed if that condition is not met.

```
if (condition) {
        statement(s);
} else {
        statement(s)2;
```

The important thing to remember when using this construct is that unless the condition is explicitly met, the else statement will be executed. In other words, the statements after the else constitute the default action while the statements after the if condition are the exception to the rule. You can now rewrite the numbers.php page incorporating else into the if statement.

Logical

Logical operators help you create reasonable constructs-statements that have a value of either TRUE or FALSE. In PHP, a condition is TRUE if it is simply a variable name and that variable has a value that is not zero (as you've seen already), such as

```
 $Variable = 5;
if ($Variable) { ...
```

A condition is also TRUE if it makes logical sense, like:

```
if (5 >= 3) {...
```

Your condition will be FALSE if it refers to a variable and that variable has no value or if you have created an illogical construct. The following conditional will always be false:

```
if (5 <= 3) { ...
```

To go beyond simple one-part conditions, PHP supports six types of logical operators: two versions of and (AND or &&); two versions of or (OR or || - a character called the pipe put together twice); not (NOT); and or not (XOR). Using parentheses and logical operators, you can create even more in-depth conditionals for your "if" conditionals. For an AND conditional, every conjoined part must be TRUE. With OR, one subsection must be TRUE to render the whole condition TRUE. These conditionals are TRUE:

if ((5 <= 3) OR (5 >= 3)) { ... if ((5 > 3) AND (5 < 10)) { ...

These c0onditionals are FALSE:

if ((5 != 5) AND (5 > 3)) { ...
if ((5 != 5) OR (5 < 3)) { ...

```
<HTML ><BODY>
<FORM ACTION="script6.php" METHOD=POST>
Quantity <INPUT TYPE=TEXT NAME="Quantity" SIZE=10><BR>
Discount <INPUT TYPE=TEXT NAME="Discount" SIZE=10><BR>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit">
</FORM></BODY></HTML>
```

Script6. HTML for capturing the Quantity and Discount

```
<HTML>
<HEAD><TITLE>Conditionals</TITLE></HEAD>
<BODY>
<?php
/* $Quantity must be passed to this page from a form or via the URL. $Discount is optional. */
$Cost = 20.00;
$Tax = 0.06;
if ($Quantity and $Discount) {
        $Quantity = abs($Quantity);
        $Discount = abs($Discount);
        $Tax++; // $Tax is now worth 1.06.
        $TotalCost = ($Cost * $Quantity);
        if ((($TotalCost < 50) AND $Discount) ) {
                print ("Your \$$Discount discount will not apply because the total value of the sale is under $50!\n<P>");
        }
        if ($TotalCost >= 50) {
                $TotalCost = $TotalCost-$Discount;
        }
        $TotalCost = $TotalCost * $Tax;
        $Payments = round ($TotalCost, 2) / 12; // Print the results.
        print ("You requested to purchase $Quantity widget(s) at \$$Cost each.\n<P>");
        print ("The total with tax, minus your \$$Discount, comes to $");
        printf ("%01.2f", $TotalCost);
        print (".\n<P>You may purchase the widget(s)in 12 monthly installments of $");
        printf ("%01.2f", $Payments);
        print (" each.\n<P>");
} else {
        print ("Please make sure that you have entered both a quantity and an applicable discount and then resubmit.\n");
}
?>
</BODY>
</HTML>
```

Script6. In this script the logical operator AND establishes a specific condition under which the message will be printed. The AND requires that both sub-conditions are TRUE in order for the whole condition to be TRUE.

Using Elseif

Similar to the if-else conditional is the if-elseif (or if-elseif-else). It acts like a running if statement and can be expanded to whatever length you require.

```
if (conditional) {
        statement(s);
} elseif (conditional2) {
        statement(s)2;
}
```

Here's another example:

```
if (conditional) {
        statement(s);
} elseif (conditional2) {
        statement(s)2;
} else {
        statement(s)3;
}
```

```
<html>
<body>
<FORM ACTION="script7.php" METHOD=POST>
User Name <INPUT TYPE=TEXT NAME="Username" SIZE=20><BR>
<INPUT TYPE=SUBMIT NAME="SUBMIT" VALUE="Submit">
</FORM>
</body>
</html>
```

Script7. HTML form to capture the $Username

```
<HTML>
<HEAD>
<TITLE>If-elseif Conditionals</TITLE>
<BODY>
<?php
if ($Username) {
        print ("Good ");
        if (date("A") == "AM") {
        print ("morning, ");
        } elseif (( date("H") >= 12) and ( date("H") < 18 )) {
                print ("afternoon, ");
                } else { print ("evening, ");
                } // Close the date if.
                print ("$Username"); print ("!\n");
        } else {
        print ("Please log in.\n");
} // Close the username if.
?>
</BODY>
</HTML>
```

Script7. It utilizes an if-else-if conditional and the date() function to write a customized greeting to the user.

The Switch Conditional

Once you get to the point where you have very elaborate if-elseif-else conditionals, you may find that it saves you time and clarifies your programming to use a switch conditional instead. The switch conditional takes only one possible condition and that is the value of a variable.

```
switch ($Variable) {
        case "values":
                statement(s)1;
                break;
        case "value2":
                statement(s)2;
                break;
        default:
        statement(s)3;
                break;
}
```

It is critical that you comprehend how a switch conditional works. Starting at the beginning, once PHP finds the case that matches the value of the set variable, it will continue to execute statements until it either comes to the end of the switch conditional (the closing curly brace) or hits a break statement, at which point it will exit the switch construct. Thus, it is imperative that you close every case (and even the default case, for consistency sake) with a break. This above switch conditional is somewhat like a rewrite of this:

```
if ($Variable == "value1") {
        statement(s)1;
} elseif ($Variable=="vatue2") {
        statement(s)2;
} else {
        statement(s)3;
}
```

I'll explain: because the switch conditional uses the value of $Variable as its condition, it will first check to see if $Variable is equal to value1 and, if so, will execute statement(s)1. If not, it will check to see if $Variable is equal to value2, and, if so, will execute statement(s)2. If neither condition is met, the default action of the switch condition is to execute statement(s)3.

In the next section, The While Loop, you'll use switch in connection with a loop to create an HTML form that tells you how many days are in a month but to demonstrate switch's capabilities here, you'll write a simple script that prints a message based upon what choice the user selects in an HTML form.

```
<HTML><HEAD><TITLE>HTML Contact Form</TITLE></HEAD><BODY>
<FORM ACTION="script8.php" METHOD=POST>
First Name <INPUT TYPE=TEXT NAME="FirstName" SIZE=Z0><BR>
Last Name <INPUT TYPE=TEXT NAME="LastName" SIZE=20><BR>
How would you prefer to be contacted:
<SELECT NAME="ContactHow">
<OPTION VALUE="">Select One:</OPTION>
<OPTION VALUE="Telephone">Telephone</OPTION>
<OPTION VALUE="Mail">Mail</OPTION>
<OPTION VALUE="E-Mail">E-Mail</OPTION>
<OPTION VALUE="Fax">Fax</OPTION>
</SELECT><BR>
Comments <TEXTAREA NAME="Comments" ROWS=5 COLS=40></TEXTAREA><BR>
<INPUT TYPE=SUBMIT NAME="SUBMIT"
VALUE="Submit!">
</FORM></BODY></HTML>
```

Script8. HTML form with pull down options.


```
<HTML><HEAD><TITLE>Contact Information Request</TITLE><BODY>
<FORM ACTION="script8.php" METHOD=POST>
<?php
// Pass on the received values using HIDDEN INPUT types.
print ("<INPUT TYPE=HIDDEN NAME=\"FirstName\" VALUE=\"$FirstName\">\n");
print ("<INPUT TYPE=HIDDEN NAME=\"LastName\" VALUE=\"$LastName\">\n");
print ("<INPUT TYPE=HIDDEN NAME=\"Comments\" VALUE=\"$Comments\">\n");
print ("<INPUT TYPE=HIDDEN NAME=\"ContactHow\" VALUE=\"$ContactHow\">\n");
switch ($ContactHow) {
        case "Telephone":
                print("<B>Please enter a daytime phone number where you can be reached:</B><BR>\n");
                print ("<INPUT TYPE=TEXT NAME= \"Telephone\" SIZE=10><BR>\n");
                print ("<INPUT TYPE=SUBMIT NAME= SUBMIT VALUE=\"Continue\">\n");
                break;
        case "Mail":
                print("<B>Please enter your complete mailing address: </B><BR>\n");
                print ("<TEXTAREA NAME=\"MailAddress\" ROWS=S COLS=40></TEXTAREA><BR>\n");
                print ("<INPUT TYPE=SUBMIT NAME= SUBMIT VALUE=\"Continue\">\n");
                break;
        case "E-Mail":
                print("<B>Please enter your E-Mail address:</B><BR>\n");
                print ("<INPUT TYPE=TEXT NAME= \"E-Mail\" SIZE=40><BR>\n");
                print ("<INPUT TYPE=SUBMIT NAME= SUBMIT VALUE=\"Continue\">\n");
                break;
        case "Fax":
                print("<B>Please enter your Fax number:</B><BR>\n");
                print ("<INPUT TYPE=TEXT NAME= \"Fax\" SIZE=10><BR>\n");
                print ("<INPUT TYPE=SUBMIT NAME= SUBMIT VALUE=\"Continue\">\n");
                break;
        default:
                print("<B>Please go back and select how you would prefer to be contacted!</B><BR>\n");
                break;
}
?>
</FORM></BODY></HTML>
```

Script8. The "switch" conditional in this script uses the value of $ContactHow to determine what to request from the user: telephone number, fax number, E-mail address, or mailing address. Hidden input types are also utilized to pass along other existing values.

The While Loop

As I suggested earlier in this chapter, loops are used to execute a section of code repeatedly. You may want to create a pull-down menu consisting of the days of the month (print the numbers 1 through 31). You might want to print out each value of an array. For either of these cases, and for many more, you'll want to use a loop. The first of the two types of loops that exist in PHP-the while loop-is designed to continue working as long as the condition you establish is TRUE. It will check the value of the condition prior cycle. Once the condition becomes FALSE, the while loop is exited.

```
while (condition) {
        statement(s);
}
```

To demonstrate the while loop, you'll create a script that dynamically generates a date pull-down menu (month, day, year) for an HTML form. While the form itself won't do anything as is, you'll be able to see how you can use PHP to improve upon and expedite the creation of a standard HTML form element.

```
<HTML><HEAD><TITLE>Select Menu</TITLE><BODY>
<?php
$Year = date ("Y"); // Create the form.
print ("<FORM ACTION=\"$PHP_SELF\" METHOD=POST>\n"); // Create the month pull-down menu.
print ("Select a month:<BR>\n");
print ("<SELECT NAME=Month><OPTION> Choose One</OPTION>\n");
print ("<OPTION VALUE=January>January </OPTION>\n");
print ("<OPTION VALUE=February>February </OPTION>\n");
print ("<OPTION VALUE=March>March </OPTION>\n");
print ("<OPTION VALUE=April>April </OPTION>\n");
print ("<OPTION VALUE=May>May </OPTION>\n");
print ("<OPTION VALUE=June>June </OPTION>\n");
print ("<OPTION VALUE=July>July /OPTION>\n");
print ("<OPTION VALUE=August>August </OPTION>\n");
print ("<OPTION VALUE=September>September </OPTION>\n");
print ("<OPTION VALUE=October>October </OPTION>\n");
print ("<OPTION VALUE=November>November </OPTION>\n");
print ("<OPTION VALUE=December>December </OPTION>\n");
print ("</SELECT>\n"); // Create the day pull-down menu.
print ("<P>Select a day:<BR>\n");
print ("<SELECT NAME=Day><OPTION>Choose One</OPTION>\n");
$Day = 1;
while ($Day <= 31) {
        print ("<OPTION VALUE=$Day>$Day </OPTION>\n");
        $Day++;
}
print ("</SELECT>\n"); // Create the year pull-down menu.
print ("<P>Select a year:<BR>\n");
print ("<SELECT NAME=Year><OPTION> Choose One</OPTION>\n");
$EndYear = $Year + 10;
while ($Year <= $EndYear) {
        print ("<OPTION VALUE=$Year>$Year </OPTION>\n");
        $Year++;
}
print ("</SELECT>\n");
print ("<P><INPUT TYPE=SUBMIT NAME=SUBMIT VALUE=\"Go!\"></FORM>\n");
?>
</BODY></HTML>
```

Script9. The two while loops will quickly print out all the requisite HTML to generate two of the pull-down menus. By using the date() function to base the ear loop on the current year, this script will never be outdated.

The For Loop

The for loop is designed to perform the specific statements for a determined number of iterations (unlike while, which runs until the condition is FALSE-similar, but significantly different, concepts). You normally use a dummy variable in the loop for this purpose. The for loop's syntax is more complicated than the while loop and although the uses of these loops can easily overlap, you'll find one more suited to some tasks than the other.

```
for (initial expression; condition; closing expression) {
        statement(s);
}
```

The initial expression will be executed once, the very first time the loop is called. Then the condition is used to determine whether or not to execute the statements. Finally, the closing expression will be executed after each time that the condition is found to be TRUE, but only after the statements are executed. Thus, to print out each value in an array, you would code:


```
For ($n = 0; $n < count($Array); $n++) {
        print ("$Array[$n]<BR>\n");
}
```

It may help your comprehension of the for loop syntax if I were to rewrite the $Day while loop from Script10 as a “for” loop. The original code was:

```
$Day = 1;
while ($Day <= 31) {
        print ("<OPTION VALUE=$Day>$Day</OPTION>\n");
        $Day++;
}
```

First the value of $Day was set, then the while loop stated the conditional ($Day <= 31). Finally, if TRUE, the print() statement was executed and $Day was incremented. As a for loop, that same code would look like this:

```
For ($Day = 1; $Day <= 31; $Day++) {
        print ("<OPTION VALUE=$Day>$Day</OPTION>\n");
}
```

A typical of using “for” loop is given below.

```
<HTML><HEAD><TITLE>Prime Numbers</TITLE>
<BODY>
<?php // Change this value if you want to print more primes.
for ($n = 1; $n <= 1000; $n++) {
if ( ($n == 1) OR ($n == 2) OR ($n == 3) OR ($n == 5)) {
        print("$n<BR>\n");
        } elseif (($n % 2 != 0) AND ($n % 3 != 0) AND ($n % 5 != 0)) {
                print("$n<BR>\n");
        } // Close the IF.
} // Close the FOR.
?>
</BODY>
</HTML>
```

Script10. A short script using “for” loop to print out the prime numbers between 1 and 1000.

7.6 Connecting to Database

A database is a collection of tables (tables being make up of columns and rows) that stores information. Databases are used all over the Internet. E-Commerce sites use databases to keep product specifications (such as price and color) as well as customer data, while content sites out articles and news stories into databases.

MYSQL

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network. To add, access, and process data stored in a computer database, you need a database management system such as MYSQL Server. Since computers are very good at handling large amounts of data, database management plays a central role in computing, as stand-alone utilities, or as parts of other applications.

MYSQL stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility. The tables are linked by defined relations making it possible to combine data from several tables on request. The SQL part of ``MYSQL'' stands for ``Structured Query Language'' - the most common standardized language used to access databases.

To create a database with remote client access on a MYSQL database server, we can use the following command:

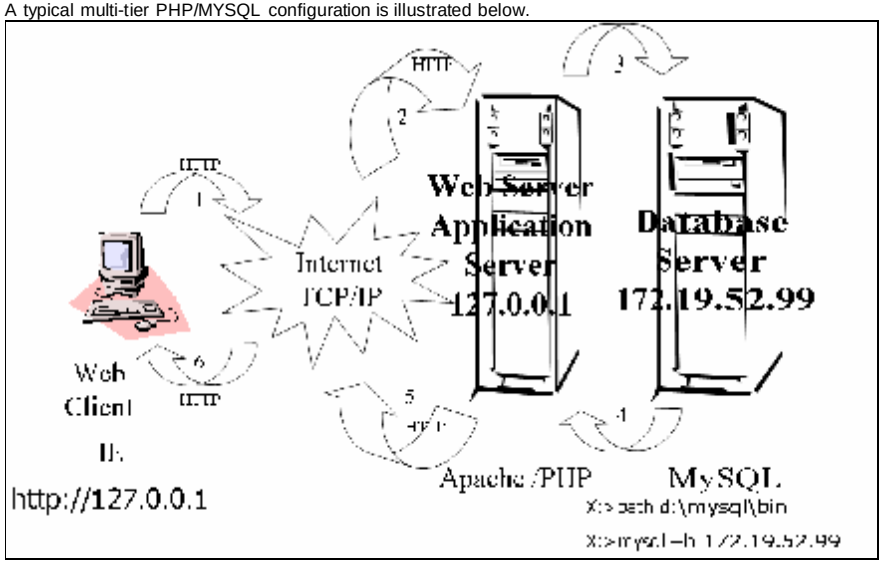
```
mysql –h hostname –u username –p
```

For example:

```
mysql –h 172.19.52.99 –u guest –p qwerty
```

It will connect the MYSQL database server with IP 172.19.52.99 and using the username "guest" with password "qwerty" to login.

A typical multi-tier PHP/MYSQL configuration is illustrated below.



Imagine now we would like to develop a user registration application. The first thing we should do is to create a database with MYSQL DML (Data modeling Language).

To create the database and table, we can use the following command:

```
Mysql>create database userlogin;
Mysql>use userlogin;
Mysql>create table userpass (username varchar(10) not null primary key, password varchar(10),age int);
```

Then we can develop the HTML and PHP program as follows.

```
<html><body>
<form action=script11.php method=post>
Login Name: <input type=text name=a_name size=10 maxlength=10><p>
Password: <input type=password name=b_word size=10 maxlength=10><p>
Age: <input type=text name=c_age size=5 maxlength=5><p
<input type=OK>
</form></body></html>
```

Script11. Capturing the "a_name", "b_word" and "c_age" inputs and use POST method to pass the content to script11.php.


```
<html><body>
<?php
mysql_connect ('172.19.52.99', 'guest', 'qwerty');
mysql_select_db ('userlogin');
if ($a_name and $b_word) {
        print ("<h1>Thanks for submission. $a_name.</h1><p>");
        $nameok = mysql_query ("SELECT * FROM userpass where username='$a_name'");
        if($row = mysql_fetch_array($nameok)){
                print ("<p><h1>Try anther name.</h1>");
                } else {
                mysql_query ("INSERT INTO userpass (username,password,age) VALUES ('$a_name',
'$b_word',$c_age)");
                $checkok = mysql_query ("SELECT * FROM userpass where username='$a_name'");
                        if ($row = mysql_fetch_array($checkok)) {
                                do {
                                print $row["username"];
                                print ("<p>");
                                print ("Your registration is successful.");
                                } while($row = mysql_fetch_array($checkok));
                                } else {print "Sorry, registration is not successful";}
                                }
        } else {
        print ("<h2>Please enter BOTH login name and password (age is optional).</h2>");
}
?>
</body></html>
```

Script11. Using PHP/MYSQL programming interface to address the database and arrange the result with if/do-while controls.

Other PHP/MYSQL functions (www.php.net)

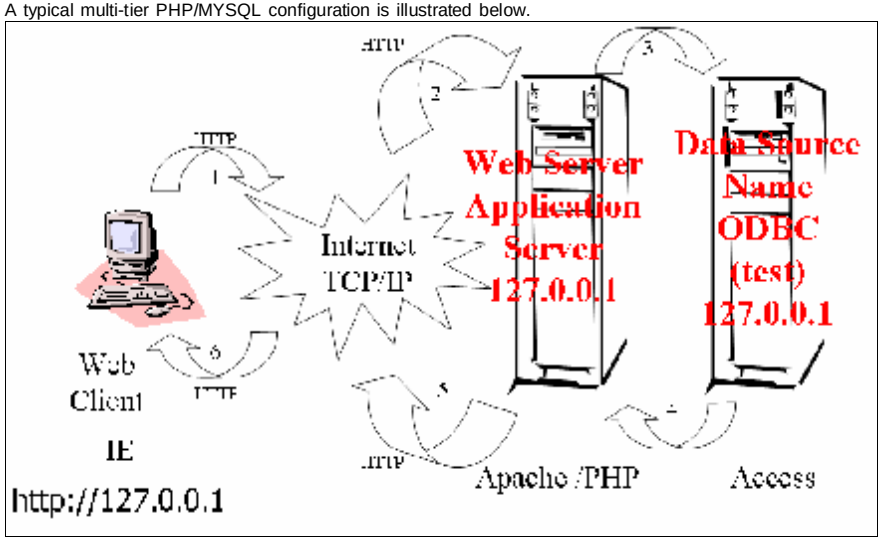
mysql_affected_rows	Get number of affected rows in previous MySQL operation
mysql_change_user	Change logged in user of the active connection
mysql_close	Close MySQL connection
mysql_connect	Open a connection to a MySQL Server
mysql_create_db	Create a MySQL database
mysql_data_seek	Move internal result pointer
mysql_db_name	Get result data
mysql_db_query	Send a MySQL query
mysql_drop_db	Drop (delete) a MySQL database
mysql_errno	Returns the numerical value of the error message from previous MySQL operation
mysql_error	Returns the text of the error message from previous MySQL operation
mysql_escape_string	Escapes a string for use in a mysql_query.
mysql_fetch_array	Fetch a result row as an associative array, a numeric array, or both.
mysql_fetch_assoc	Fetch a result row as an associative array
mysql_fetch_field	Get column information from a result and return as an object
mysql_fetch_lengths	Get the length of each output in a result
mysql_fetch_object	Fetch a result row as an object
mysql_fetch_row	Get a result row as an enumerated array
mysql_field_flags	Get the flags associated with the specified field in a result
mysql_field_name	Get the name of the specified field in a result
mysql_field_len	Returns the length of the specified field
mysql_field_seek	Set result pointer to a specified field offset
mysql_field_table	Get name of the table the specified field is in

mysql_field_type	**Get the type of the specified field in a result**
mysql_free_result	**Free result memory**
mysql_insert_id	**Get the id generated from the previous INSERT operation**
mysql_list_dbs	**List databases available on a MySQL server**
mysql_list_fields	**List MySQL result fields**
mysql_list_tables	**List tables in a MySQL database**
mysql_num_fields	**Get number of fields in result**
mysql_num_rows	**Get number of rows in result**
mysql_pconnect	**Open a persistent connection to a MySQL server**
mysql_query	**Send a MySQL query**
mysql_unbuffered_query	**Send an SQL query to MySQL, without fetching and buffering the result rows**
mysql_result	**Get result data**
mysql_select_db	**Select a MySQL database**
mysql_tablename	**Get table name of field**
mysql_get_client_info	**Get MySQL client info**
mysql_get_host_info	**Get MySQL host info**
mysql_get_proto_info	**Get MySQL protocol info**
mysql_get_server_info	**Get MySQL server info**

Microsoft ACCESS

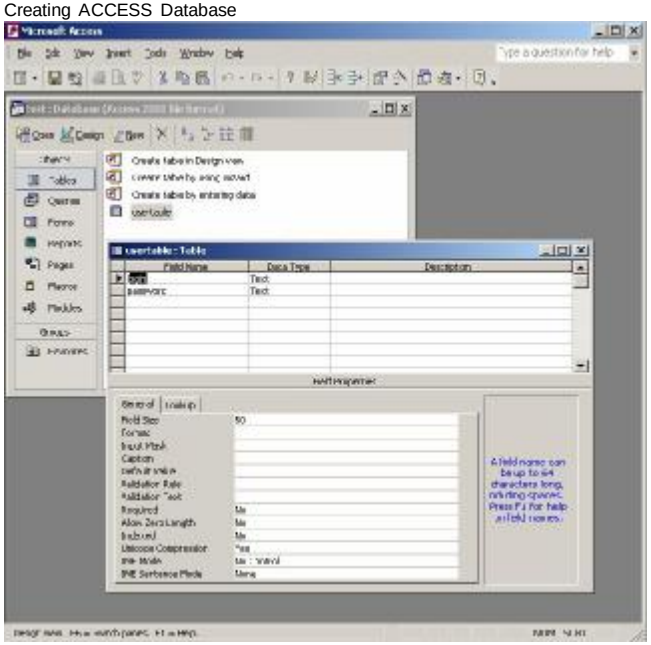
The connection of PHP/ACCESS is usually done with ODBC. Open Database Connectivity (ODBC) is a widely accepted application programming interface (API) for database access. It is based on the Call-Level Interface (CLI) specifications from X/Open and ISO/IEC for database APIs and uses Structured Query Language (SQL) as its database access language.

A typical multi-tier PHP/MYSQL configuration is illustrated below.

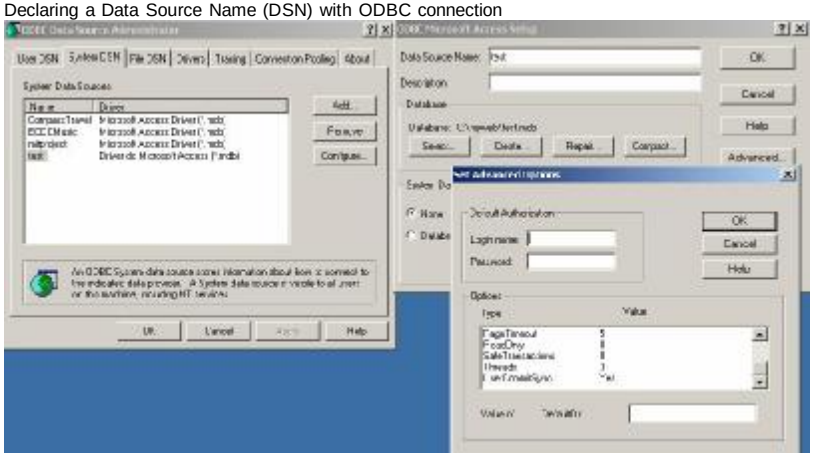


To develop a PHP/ODBC connection, the first step is to create a ACCESS database, then declare a SYSTEM DSN with administrator or power user accounts.

Creating ACCESS Database



Declaring a Data Source Name (DSN) with ODBC connection



By using ODBC, we can develop the HTML and PHP programs to interact with the database.

```
<html><head>Input username and password with PHP+ODBC</head><body>
<form action=password.php method=GET>
Username: <input type=text name=username size=25 maxlength=25>
Password: <input type=password name=userpass size=25 maxlength=25><p>
<input type=submit>
</form></body></html>
```

Script12. HTML for capturing the user input.

```
<html>
<head><title>User Name and Password Registration</title></head><body>
<title>PHP ODBC Connection</title>
<body bgcolor="#FFFFFF">
<table width="75%" border="1" cellspacing="1" cellpadding="1" bgcolor="#FFFFFF">
  <tr bgcolor="#CCFFFF">
    <td height="22"><b>User Name</b></td>
    <td height="22"><b>Password</b></td>  </tr>
<?php
    //connect to the database
    $connectionstring = odbc_connect("test", "", "");

    //SQL query
    $Query1 = "insert into usertable values ('$username','$userpass')";
    $Query2 = "SELECT * FROM usertable where login='$username'";
    //execute query
    $queryexe1 = odbc_do($connectionstring, $Query1);
    $queryexe2 = odbc_do($connectionstring, $Query2);
    //query database
    while(odbc_fetch_row($queryexe2))
    {
    $uname = odbc_result($queryexe2, 1);
    $pass = odbc_result($queryexe2, 2);
    //format results
    print ("<h1>Input successful</h1>");
    print ("<br>");
    print ("<tr>");
    print ("<td>$uname</td>");
    print ("<td>$pass</td>");
    print ("</tr>");
    }
    //disconnect from database
    odbc_close($connectionstring);
    ?>
</table>
</body>
</html>
```

Script12. Apply ODBC to interact with ACCESS database.

Other PHP/ODBC functions (www.php.net)

odbc_autocommit	Toggle autocommit behaviour
odbc_binmode	Handling of binary column data
odbc_close	Close an ODBC connection
odbc_close_all	Close all ODBC connections
odbc_commit	Commit an ODBC transaction
odbc_connect	Connect to a datasource
odbc_cursor	Get cursorname
odbc_do	Synonym for odbc_exec()
odbc_error	Get the last error code
odbc_errormsg	Get the last error message
odbc_exec	Prepare and execute a SQL statement
odbc_execute	Execute a prepared statement
odbc_fetch_into	Fetch one result row into array
odbc_fetch_row	Fetch a row
odbc_fetch_array	Fetch a result row as an associative array
odbc_next_result	Checks if multiple results are avaiable
odbc_fetch_object	Fetch a result row as an object
odbc_field_name	Get the columnname
odbc_field_num	Return column number
odbc_field_type	Datatype of a field
odbc_field_len	Get the length (precision) of a field
odbc_field_precision	Synonym for odbc_field_len()
odbc_field_scale	Get the scale of a field
odbc_free_result	Free resources associated with a result
odbc_longreadlen	Handling of LONG columns
odbc_num_fields	Number of columns in a result
odbc_pconnect	Open a persistent database connection
odbc_prepare	Prepares a statement for execution
odbc_num_rows	Number of rows in a result
odbc_result	Get result data
odbc_result_all	Print result as HTML table
odbc_rollback	Rollback a transaction
odbc_setoption	Adjust ODBC settings. Returns FALSE if an error occurs, otherwise TRUE.
odbc_tables	Get the list of table names stored in a specific data source. Returns a result identifier containing the information.
odbc_tableprivileges	Lists tables and the privileges associated with each table
odbc_columns	Lists the column names in specified tables. Returns a result identifier containing the information.
odbc_columnprivileges	Returns a result identifier that can be used to fetch a list of columns and associated privileges
odbc_gettypeinfo	Returns a result identifier containing information about data types supported by the data source.
odbc_primarykeys	Returns a result identifier that can be used to fetch the column names that comprise the primary key for a table
odbc_foreignkeys	Returns a list of foreign keys in the specified table or a list of foreign keys in other tables that refer to the primary key in the specified table
odbc_procedures	Get the list of procedures stored in a specific data source. Returns a result identifier containing the information.
odbc_procedurecolumns	Retrieve information about parameters to procedures
odbc_specialcolumns	Returns either the optimal set of columns that uniquely identifies a row in the table or columns that are automatically updated when any value in the row is updated by a transaction
odbc_statistics	Retrieve statistics about a table