# Chapter One - Server Side Scripting Basics

# Introduction

Today’s Web users expect exciting pages that are updated frequently and provide a customized experience. For them, Web sites are more like communities, to which they’ll return time and again. At the same time, Web-site administrators want sites that are easier to update and maintain, understanding that’s the only reasonable way to keep up with visitors’ expectations. For these reasons and more, PHP and MySQL have become the de facto standards for creating dynamic, database driven Web sites.

What Are Dynamic Web Sites?

Dynamic Web sites are flexible and potent creatures, more accurately described as applications than merely sites. DynamicWeb sites

* Respond to different parameters (for example, the time of day or the version of the visitor’s Web browser)
* Have a “memory,” allowing for user registration and login, e-commerce, and similar processes
* Almost always integrate HTML forms, allowing visitors to perform searches, provide feedback, and so forth
* Often have interfaces where administrators can manage the site’s content
* Are easier to maintain, upgrade, and build upon than statically made sites

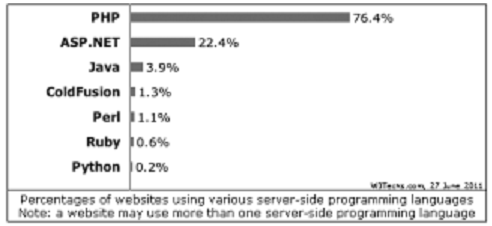
There are many technologies available for creating dynamic Web sites. The most common are ASP.NET (Active Server Pages, a Microsoft construct), JSP (Java Server Pages), ColdFusion, Ruby on Rails (a Web development framework for the Ruby programming language), and PHP. Dynamic Web sites don’t always rely on a database, but more and more of them do, particularly as excellent database applications like MySQL are available at little to no cost.

What is PHP?

PHP originally stood for “Personal Home Page” as it was created in 1994 by Rasmus Lerdorf to track the visitors to his online résumé. As its usefulness and capabilities grew (and as it started being used in more professional situations), it came to mean “PHP: Hypertext Preprocessor.” According to the official PHP Web site, found at www.php.net A, PHP is a “widely used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.”

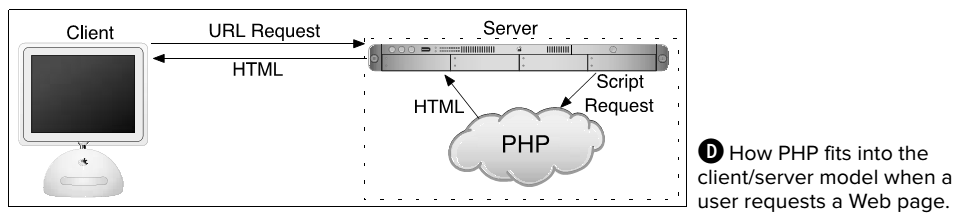
PHP is a server-side, cross-platform technology, both descriptions being important. Server side refers to the fact that everything PHP does occurs on the server. A Web server application, like Apache or Microsoft’s IIS (Internet Information Services), is required and all PHP scripts must be accessed through a URL (http://something). Its cross-platform nature means that PHP runs on most operating systems, including Windows, UNIX (and its many variants), and Macintosh. More important, the PHP scripts written on one server will normally work on another with little or no modification.

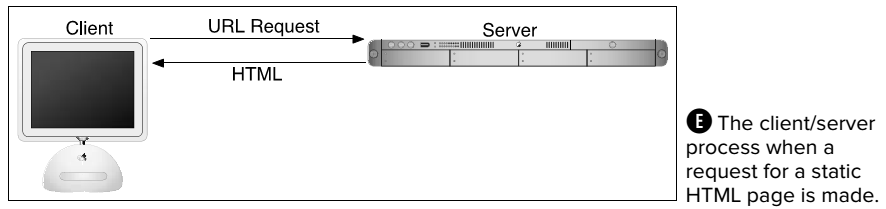
PHP has seen an exponential growth in use since its inception, and is the server-side technology of choice on over 76 percent of all Web sites. In terms of all programming languages, PHP is the fifth most popular after Java, C, C++ and C#.



How pHp works?

As previously stated, PHP is a server-side language. This means that the code you write in PHP sits on a host computer called a server. The server sends Web pages to the requesting visitors (you, the client, with your Web browser).When a visitor goes to a Web site written in PHP, the server reads the PHP code and then processes it according to its scripted directions. In the example shown in below, the PHP code tells the server to send the appropriate data—HTML code—to the Web browser, which treats the received code as it would a standard HTML page. This differs from a static HTML site where, when a request is made, the server merely sends the HTML data to the Web browser and there is no server-side interpretation occurring. Because no server-side action is required, you can run HTML pages in your Web browser without using a server at all. To the end user and the Web browser there is no perceptible difference between what **home.html** and **home.php** may look like, but how that page’s content was created will be significantly different.



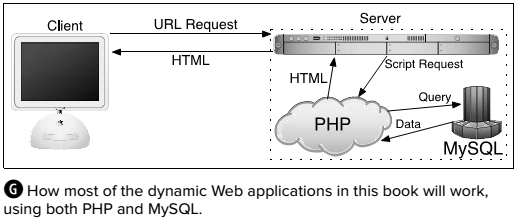


What is MySQL?

MySQL (www.mysql.com) is the world’s most popular open-source database. In fact, today MySQL is a viable competitor to the pricey goliaths such as Oracle and Microsoft’s SQL Server (and, ironically, MySQL is owned by Oracle). Like PHP, MySQL offers excellent performance, portability, and reliability, with a moderate learning curve and little to no cost.

MySQL is a database management system (DBMS) for relational databases (therefore, MySQL is an RDBMS). A database, in the simplest terms, is a collection of data, be it text, numbers, or binary files, stored and kept organized by the DBMS.

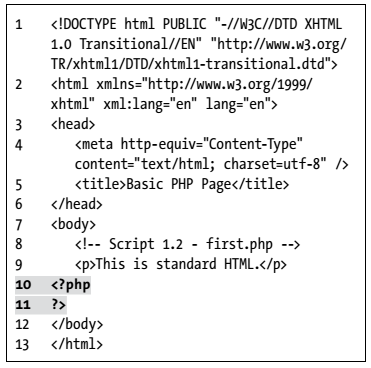
By incorporating a database into a Web application, some of the data generated by PHP can be retrieved from MySQL . This further moves the site’s content from a static (hard-coded) basis to a flexible one, flexibility being the key to a dynamic Web site.



MySQL has been known to handle databases as large as 60,000 tables with more than 5 billion rows. MySQL can work with tables as large as 8 million terabytes on some operating systems, generally a healthy 4 GB otherwise. MySQL is used by NASA and the United States Census Bureau, among many others.

Basic Syntax in PHP

PHP is an HTML-embedded scripting language, meaning that you can intermingle PHP and HTML code within the same file. So to begin programming with PHP, start with a simple Web page. Script 1.1 is an example of a no-frills, no-content XHTML Transitional document, which will be used as the foundation for most Web pages. Please also note that the template uses UTF-8 encoding, a topic discussed in the sidebar. To add PHP code to a page, place it with in PHP tags:



<?php

?>

Anything written within these tags will be treated by the Web

server as PHP, meaning the PHP interpreter will process the

code. Any text outside of the PHP tags is immediately sent to

the Web browser as regular HTML. (Because PHP is most often

used to create content displayed in the Web browser, the PHP tags are normally put somewhere within the page’s body.)

Along with placing PHP code within PHP tags, your PHP files must have a proper extension. The extension tells the server to treat the script in a special way, namely, as a PHP page. Most Web servers use **.html** for standard HTML pages and **.php** for PHP files.

To make a basic pHpscript:

1. Create a new document in your text editor or IDE, to be named first.php
2. Create a basic HTML document
3. Before the closing bodytag, insert the PHP tags:

<?php

?>

1. Save the file as first.php.
2. Place the file in the proper directory of your Web server (localhost, webserver or IIS).
3. Run first.php in your Web browser (access them via a URL e.g. localhost/first.php).

Sending Data to the Web Browser

To create dynamic website with PHP, you must know how to send data to the Web browser. PHP has a number of built-in functions for this purpose, the most common being echo and print. I personally tend to favor echo: You could use print instead, if you prefer (the name more obviously indicates what it does):

print 'Hello, world!';

print "What's new?";

echo 'Hello, world!';

echo "What's new?";

What can PHP do?

* PHP can generate dynamic page content
* PHP can create, open, read, write, and close files on the server
* PHP can collect form data
* PHP can send and receive cookies
* PHP can add, delete, modify data in your database
* PHP can restrict users to access some pages on your website
* PHP can encrypt data

With PHP we are not limited to output HTML. We can output images, PDF files, and even flash movies. We can also output any text, such as XHTML and XML.

Why PHP?

* PHP runs on different platforms (Windows, Linux, UNIX, Mac OSX, etc.)
* PHP is compatible with almost all servers used today (Apache,IIS, etc.)
* PHP has support for a wide range of databases
* PHP is free. Download it from the official PHP resource: www.php.net
* PHP is easy to learn and runs efficiently on the server side

Writing Comments in PHP

<! DOCTYPE html>

PHP Variables

Variables are "containers" for storing information:

<html>

<body>

<?php

Example

<?php

$x=5;

$y=6;

$z=$x+$y;

echo $z;

?>

Much Like Algebra x=5, y=6, z=x+y

In algebra we use letters (like x) to hold values (like 5).

From the expression z=x+y above, we can calculate the value of z to be 11

//This is a PHP comment line

/\*

This is a PHP comment

block

\*/

?>

</body>

</html>

Rules for PHP variables:

* A variable starts with the $ sign, followed by the name of the variable
* A variable name must begin with a letter or the underscore character
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* A variable name should not contain spaces Variable names are case sensitive ($y and $Y are two different variables)

Both PHP statements and PHP variables are case-sensitive.

Creating (Declaring) PHP Variables

PHP has no command for declaring a variable. A variable is created the moment you first assign a value to it:

$txt="Hello world!"; or $x=5;

After the execution of the statements above, the variable txt will hold the value Hello world!, and the variable x will hold the value 5.

**Note**: When we assign a text value to a variable, put quotes around the value.

PHP is a Loosely Typed Language

In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value. In a strongly typed programming language, we will have to declare (define) the type and name of the variable before using it.

PHP Variable Scopes

The scope of a variable is the part of the script where the variable can be referenced/used.

PHP has four different variable scopes:

1. Local
2. global
3. static
4. parameter
5. Local Scope

A variable declared within a PHP function is local and can only be accessed within that function:

Example

<?php

The script above will not produce any output because the echo statement refers to the local scope variable $x, which has not been assigned a value within this scope.

We can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

Local variables are deleted as soon as the function is completed.

$x=5; // global scope

function myTest()

{

echo $x; // local scope

}

myTest();

?>

1. Global Scope

A variable that is defined outside of any function, has a global scope. Global variables can be accessed from any part of the script, EXCEPT from within a function.

To access a global variable from within a function, use the **global** keyword:

PHP also stores all global variables in an array called **$GLOBALS[index].**

The index holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

<?php

$x=5;

$y=10;

function myTest()

{

$GLOBALS['y']=$GLOBALS['x']+$GLOBALS['y'];

}

myTest();

echo $y;

?>

Example

<?php

$x=5; // global scope

$y=10; // global scope

function myTest()

{

global $x,$y;

$y=$x+$y;

}

myTest();

echo $y; // outputs 15

?> \*\* Note: how global key word is used.

1. Static Scope

When a function is completed, all of its variables are normally deleted. However, sometimes you want a local variable to not be deleted. To do this, use the static keyword when you first declare the variable:

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

Example

<?php

function myTest()

4. Parameter Scope

A parameter is a local variable whose value is passed to the function by the calling code.

Parameters are declared in a parameter list as part of the function declaration:

<?php

function myTest($x) \* Note: Parameters are also called arguments

{

echo $x;

}

myTest(5);

?>

{

static $x=0;

echo $x;

$x++;

}

myTest();

myTest();

myTest();

?>

PHP String Variables

A string variable is used to store and manipulate text.

String Variables in PHP

String variables are used for values that contain characters. After we have created a string variable we can manipulate it. A string can be used directly in a function or it can be stored in a variable. In the example below, we create a string variable called txt, then we assign the text "Hello world!" to it. Then we write the value of the txt variable to the output:

Example

☺ Note: When we assign a text value to a variable, remember to put single or double quotes around the value.

<?php

$txt="Hello world!";

echo $txt;

?>

The following are some commonly used functions and operators to manipulate strings.

The PHP Concatenation Operator

There is only one string operator in PHP.

The concatenation operator (.) is used to join two string values together. The example below shows how to concatenate two string variables together:

Example

The output of the code above will be:

Hello world! What a nice day!

**Tip**: In the code above we have used the concatenation operator two times. This is because we wanted to insert a white space between the two strings.

<?php

$txt1="Hello world!";

$txt2="What a nice day!";

echo $txt1 . " " . $txt2;

?>

The PHP strlen( ) function

Sometimes it is useful to know the length of a string value.

The strlen( ) function returns the length of a string, in characters. The example below returns the length of the string "Hello world!":

The output of the code above will be: 12

Tip: strlen( ) is often used in loops or other functions, when it is important to know when

a string ends. (i.e. in a loop, we might want to stop the loop after the last character in a

string).

Example

<?php

echo strlen("Hello world!");

?>

The PHP strpos ( ) function

The strpos( ) function is used to search for a character or a specific text within a string. If a match is found, it will return the character position of the first match. If no match is found, it will return FALSE. The example below searches for the text "world" in the string "Hello world!":

The output of the code above will be: **6**

Tip: The position of the string "world" in the example above is 6. The reason that it is 6 (and not 7), is that the first character position in the string is 0, and not 1.

Example

<?php

echo strpos("Hello world!","world");

?>

PHP Operators

The assignment operator = is used to assign values to variables in PHP.

The arithmetic operator + is used to add values together in PHP.

PHP Arithmetic Operators

Operator Name Description Example Result

x+y Addition Sum of x and y 2+2 4

x-y Subtraction Difference of x and y 5-2 3

x\*y Multiplication Product of x and y 5\*2 10

x/y Division Quotient of x and y 15/5 3

x%y Modulus Remainder of x divided by y 10 % 8 2

-x Negation Opposite of x -2 2

a.b Concatenation Concatenate two strings “hi”.”dear” hi dear

PHP Assignment Operators

The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the expression on the right. That is, the value of "$x = 5" is 5.

Assignment Same as... Description

x=y x=y the left operand gets set to the value of the expression on the right

x += y x=x+y Addition

x -= y x=x-y Subtraction

x \*= y x=x\*y Multiplication

x /= y x=x/y Division

x %= y x=x%y Modulus

a .= b a=a.b Concatenate two strings

PHP Incrementing/Decrementing Operators

Operator Name Description

++ x Pre-increment Increments x by one, then returns x

x ++ Post-increment Returns x, then increments x by one

-- x Pre-decrement Decrements x by one, then returns x

x -- Post-decrement Returns x, then decrements x by one

PHP Comparison Operators

Comparison operators allows us to compare two values:

Operator Name Description Example

x == y Equal True if x is equal to y 5==8 returns false

x === y Identical True if x is equal to y, they are of same type 5==="5" returns false

x!= y Not equal True if x is not equal to y 5!=8 returns true

x <> y Not equal True if x is not equal to y 5<>8 returns true

x !== y Not identical True if x is not equal to y,or they are not of same type 5!=="5" returns true

x>y Greater than True if x is greater than y 5>8 returns false

x<y Less than True if x is less than y 5<8 returns true

x >= y Greater or equal True if x is greater than or equal to y 5>=8 returns false

x <= y less than or equal True if x is less than or equal to y 5<=8 returns true

PHP Logical Operators

Operator Name Description Example

x and y And True if both x and y are true x=6,y=3 (x < 10 and y > 1) returns true

x or y Or True if either or both x and y are true x=6,y=3, (x==6 or y==5) returns true

x xor y Xor True if either x or y is true, but not both x=6,y=3, (x==6 xor y==3) returns false

x && y And True if both x and y are true x=6,y=3, (x < 10 && y > 1) returns true

x || y Or True if either or both x and y are true x=6,y=3, (x==5 || y==5) returns false

!x Not True if x is not true x=6,y=3, !(x==y) returns true

PHP Conditional Statements

Very often when we write code, we want to perform different actions for different decisions. We can use conditional statements in our code to do this.

In PHP we have the following conditional statements:

* **if statement** - executes some code only if a specified condition is true
* **if...else statement** - executes some code if a condition is true and another code if the condition is false
* **if...else if....else statement** - selects one of several blocks of code to be executed
* **switch statement** - selects one of many blocks of code to be executed

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else statement.

Syntax

if (condition)

code to be executed if condition is true;

else

code to be executed if condition is false;

Example

The following example will output "Have a nice weekend!" if the current day is Friday, otherwise it will output "Have a nice day!":

<html>

If more than one line should be executed if a condition is true/false, the lines should be enclosed within curly braces:

<?php

$d=date("D");

if ($d=="Fri")

{

echo "Hello!<br />";

echo "Have a nice weekend!";

echo "See you on Monday!";

}

?>

<body>

<?php

$d=date("D");

if ($d=="Fri")

echo "Have a nice weekend!";

else

echo "Have a nice day!";

?>

</body>

</html>

The Else… If Statement

If you want to execute some code if one of several conditions are true use the elseif statement

Syntax

if (condition)

code to be executed if condition is true;

elseif (condition)

code to be executed if condition is true;

else

code to be executed if condition is false;

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and

"Have a nice Sunday!" if the current day is Sunday. Otherwise it will output "Have a nice day!":

<?php

\*\* Note: no space between else and if.

So in this we actually have 3 option to go based on the day today.

$d=date("D");

if ($d=="Fri")

echo "Have a nice weekend!";

elseif ($d=="Sun")

echo "Have a nice Sunday!";

else

echo "Have a nice day!";

?>

PHP Switch Statement

The Switch statement in PHP is used to perform one of several different actions based on one of several different conditions. If you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..else code.

Syntax

switch (expression)

{

case label1:

code to be executed if expression = label1;

break;

case label2:

code to be executed if expression = label2;

This is how it works:

* A single expression (most often a variable) is evaluated once
* The value of the expression is compared with the values for each case in the structure
* If there is a match, the code associated with that case is executed
* After a code is executed, **break** is used to stop the code from running into the next case
* The default statement is used if none of the cases are true

break;

default:

code to be executed

if expression is different

from both label1 and label2;

}

Example

<?php

switch ($x)

{

If you assign $x as 5 i.e $x=5;

The output will be

No number between 1 and 3

case 1:

echo "Number 1";

break;

case 2:

echo "Number 2";

break;

case 3:

echo "Number 3";

break;

default:

echo "No number between 1 and 3";

}

?>

Looping

Bounded loops versus unbounded loops

A **bounded** loop executes a fixed number of times —you can tell by looking at the code how many times the loop will iterate, and the language guarantees that it won’t loop more times than that. An **unbounded** loop repeats until some condition becomes true (or false), and that condition is dependent on the action of the code within the loop. Bounded loops are predictable, whereas unbounded loops can be as tricky as you like.

PHP doesn’t actually have any constructs specifically for bounded loops —**while, do-while, and for** are all unbounded constructs —but an unbounded loop can do anything a bounded loop can do.

While

The simplest PHP looping construct is while, which has the following syntax:

while (condition)

statement

The while loop evaluates the condition expression as a Boolean —if it is true, it executes statement and then starts again by evaluating condition. If the condition is false, the while loop terminates. Of course, just as with if, statement may be a single statement or it may be a brace-enclosed block. The body of a while loop may not execute even once, as in:

while (FALSE)

print(“This will never print.<BR>”);

Or it may execute forever, as in this code snippet:

while (TRUE)

print(“All work and no play makes Jack a dull boy.<BR>”);

or it may execute a predictable number of times, as in:

$count = 1;

while ($count <= 10)

{

print(“count is $count<BR>”);

$count = $count + 1;

}

which will print exactly 10 lines.

Do-while

The do-while construct is similar to while, except that the test happens at the end of the loop. The syntax is:

do statement

while (expression);

The statement is executed once, and then the expression is evaluated. If the expression is true, the statement is repeated until the expression becomes false. The only practical difference between whileand do-whileis that the latter will always execute its statement at least once.For example:

$count = 45;

do

{

print(“count is $count<BR>”);

$count = $count + 1;

}

while ($count <= 10)

This prints the single line:

Output will be 45.

For Loop

The most complicated looping construct is for, which has the following syntax:

for (initial-expression;termination-check; loop-end-expression)

statement;

It is also legal to include more than one of each kind of for clause, separated by commas.

The termination-check will be considered to be true if any of its sub clauses are true; it is like an ‘or’ test. For example, the following statement:

Would give the browser output:

1, 1, 1

2, 3, 4

3, 5, 7

for ($x = 1, $y = 1, $z = 1; //initial expressions

$y < 10, $z < 10; // termination checks

$x = $x + 1, $y = $y + 2, // loop-end expressions

$z = $z + 3)

print(“$x, $y, $z<BR>”);

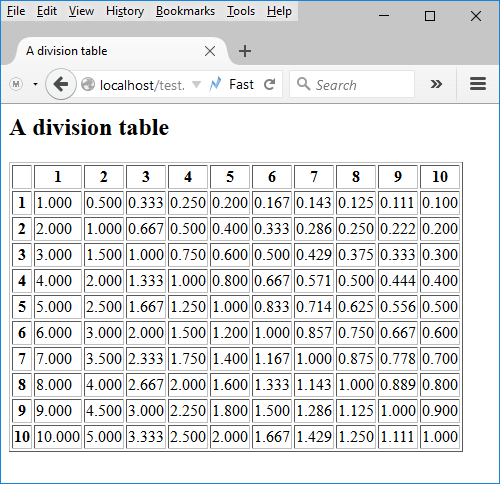
**Looping examples**create a Division Table using for loop.

<HTML>

<BODY>

<TABLE BORDER=1>

<?php



\*\* The output would look like this.

$start\_num = 1;

$end\_num = 10;

print("<TR>");

print("<TH> </TH>");

for ($count\_1 = $start\_num;

$count\_1 <= $end\_num;

$count\_1++)

print("<TH>$count\_1</TH>");

print("</TR>");

for ($count\_1 = $start\_num;

$count\_1 <= $end\_num;

$count\_1++)

{

print("<TR><TH>$count\_1</TH>");

for ($count\_2 = $start\_num;

$count\_2 <= $end\_num;

$count\_2++)

{

$result = $count\_1 / $count\_2;

printf("<TD>%.3f</TD>", $result);

}

print("</TR>\n");

}

?>

</TABLE>

</BODY> </HTML>

PHP Arrays

An array can store one or more values in a single variable name.

What is an array?

When working with PHP, sooner or later, you might want to create many similar variables. Instead of having many similar variables, you can store the data as elements in an array. Each element in the array has its own ID so that it can be easily accessed. There are three different kind of arrays:

* Numeric array- An array with a numeric ID key
* Associative array- An array where each ID key is associated with a value
* Multidimensional array- An array containing one or more arrays

Numeric Arrays

A numeric array stores each element with a numeric ID key. There are different ways to create a numeric array.

Example 1

In this example the ID key is automatically assigned:

$names = array("Tsegaye","Abebe","Joe");

Example 2

In this example we assign the ID key manually:

$names[0] = "Tsegaye";

$names[1] = "Abebe";

$names[2] = "Joe";

The ID keys can be used in a script:

<?php

$names[0] = "Tsegaye";

$names[1] = "Abebe";

$names[2] = "Joe";

echo $names[1] . " and " . $names[2] . " are ". $names[0] . "'s neighbors";

?>

Output: Abebe and Joe are Tsegaye’s neighbors.

Associative Arrays

When storing data about specific named values, a numerical array is not always the best way to do it.

With associative arrays we can use the values as keys and assign values to them.

Example 1

In this example we use an array to assign ages to the different persons:

$ages = array("Peter"=>32, "Quagmire"=>30, "Joe"=>34);

Example 2

This example is the same as example 1, but shows a different way of creating the array:

$ages['Peter'] = "32";

$ages['Quagmire'] = "30";

The code above will output:

Peter is 32 years old.

$ages['Joe'] = "34";

The ID keys can be used in a script:

<?php

$ages['Peter'] = "32";

$ages['Quagmire'] = "30";

$ages['Joe'] = "34";

echo "Peter is " . $ages['Peter'] . " years old.";

?>

Multidimensional Arrays

In a multidimensional array, each element in the main array can also be an array. And each element in the sub-array can be an array, and so on.

Example

In this example we create a multidimensional array, with automatically assigned ID keys:

$families = array

("Griffin"=>array("Peter", "Lois", "Megan"),"Quagmire"=>array("Glenn"), "Brown"=>array("Cleveland", "Loretta", "Junior"));

The array above would look like this if written to the output:

Array

([Griffin] => Array([0] => Peter[1] => Lois[2] => Megan) [Quagmire] => Array([0] => Glenn) [Brown] => Array([0] => Cleveland[1] => Loretta[2] => Junior) )

The foreach Statement

The foreach statement is used to loop through arrays.

For every loop, the value of the current array element is assigned to $value (and the array pointer is moved by one) - so on the next loop, you'll be looking at the next element.

Syntax

foreach (array asvalue)

{

code to be executed;

}

Example

The following example demonstrates a loop that will print the values of the given array:

<?php

$arr=array("one", "two", "three");

foreach ($arr as $value)

{

echo "Value: " . $value . "<br />";

}

?>

PHP Functions

The real power of PHP comes from its functions. In PHP - there are more than 700 built-in functions available.

For a reference and examples of the built-in functions, please visit our PHP Reference.

Create a PHP Function

A function is a block of code that can be executed whenever we need it.

Creating PHP functions:

* All functions start with the word "function()"
* Name the function - It should be possible to understand what the function does by its name. The name can start with a letter or underscore (not a number)
* Add a "{" - The function code starts after the opening curly brace
* Insert the function code
* Add a "}" - The function is finished by a closing curly brace.

Example

A simple function that writes my name when it is called:

<?php

function writeMyName()

{

echo "Tsegaye Andargie";

}

writeMyName();

?>

PHP Functions - Adding parameters

Our first function (writeMyName()) is a very simple function. It only writes a static string.

To add more functionality to a function, we can add parameters. A parameter is just like a variable.

You may have noticed the parentheses after the function name, like: writeMyName(). The parameters are specified inside the parentheses.

Example 1

The following example will write different first names, but the same last name:

<?php

function writeMyName($fname)

We can call a function as many times as we want.

The output of the program will be

My name is Tsegaye

My name is Andargie

{

echo $fname . "<br />";

}

echo "My name is ";

writeMyName("Tsegaye");

echo "My name is ";

writeMyName("Andargie");

?>

Example 2

The following function has two parameters:

<?php

function writeMyName($fname,$punctuation)

{

echo $fname . " Refsnes" . $punctuation . "<br />";

The output of the code above will be:

My name is Kai Jim Refsnes.

My name is Hege Refsnes!

My name is Ståle Refsnes...

}

echo "My name is ";

writeMyName("Kai Jim",".");

echo "My name is ";

writeMyName("Hege","!");

echo "My name is ";

writeMyName("Ståle","...");

?>

PHP Functions - Return values]

Functions can also be used to return values.

Example

<?php

The output of the code above will be:

1 + 16 = 17

function add($x,$y)

{

$total = $x + $y;

return $total;

}

echo "1 + 16 = " . add(1,16)

?>

# Chapter Two- Forms

The PHP superglobals $\_GET and $\_POST are used to collect form-data. The example below displays a simple HTML form with two input fields and a submit button:

<html>  
<body>  
<form action="welcome.php" method="post">  
Name: <input type="text" name="name"><br>  
E-mail: <input type="text" name="email"><br>  
<input type="submit">  
</form>  
</body>  
</html>

When the user fills out the form above and clicks the submit button, the form data is sent for processing to a PHP file named "welcome.php". The form data is sent with the HTTP POST method.

To display the submitted data you could simply echo all the variables. The "welcome.php" looks like this:

<html>  
<body>  
Welcome <?php echo $\_POST["name"]; ?><br>  
Your email address is: <?php echo $\_POST["email"]; ?>  
</body>  
</html>

The output could be something like this:

Welcome John  
Your email address is john.doe@example.com

The same result could also be achieved using the HTTP GET method:

<html>  
<body>  
<form action="welcome\_get.php" method="get">  
Name: <input type="text" name="name"><br>  
E-mail: <input type="text" name="email"><br>  
<input type="submit">  
</form>  
</body>  
</html>

and "welcome\_get.php" looks like this:

<html>  
<body>  
Welcome <?php echo $\_GET["name"]; ?><br>  
Your email address is: <?php echo $\_GET["email"]; ?>  
</body>  
</html>

## PHP $\_REQUEST

PHP $\_REQUEST is used to collect data after submitting an HTML form using get or post method. e.g.

<html>  
<body>  
<form method="post" action="<?php echo $\_SERVER['PHP\_SELF'];?>">  
  Name: <input type="text" name="fname">  
  <input type="submit">  
</form>  
<?php  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
    // collect value of input field  
    $name = $\_REQUEST['fname'];  
    if (empty($name)) {  
        echo "Name is empty";  
    } else {  
        echo $name;  
    }  
}  
?>  
</body>  
</html>

## GET vs. POST

Both GET and POST create an array (e.g. array( key => value, key2 => value2, key3 => value3, ...)). This array holds key/value pairs, where keys are the names of the form controls and values are the input data from the user.

Both GET and POST are treated as $\_GET and $\_POST. These are superglobals, which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.

$\_GET is an array of variables passed to the current script via the URL parameters.

$\_POST is an array of variables passed to the current script via the HTTP POST method.

## When to use GET?

Information sent from a form with the GET method is **visible to everyone** (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

**Note:** GET should NEVER be used for sending passwords or other sensitive information!

## When to use POST?

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send.

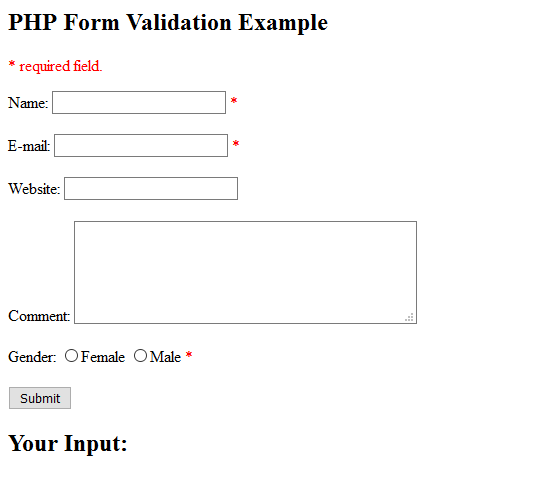
Moreover POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

|  |  |
| --- | --- |
| Note | **Developers prefer POST for sending form data.** |

# PHP - Form Validation

Proper validation of form data is important to protect your form from hackers and spammers!



The validation rules for the form above are as follows:

|  |  |
| --- | --- |
| Field | Validation Rules |
| Name | Required. + Must only contain letters and whitespace |
| E-mail | Required. + Must contain a valid email address (with @ and .) |
| Website | Optional. If present, it must contain a valid URL |
| Comment | Optional. Multi-line input field (textarea) |
| Gender | Required. Must select one |

First we will look at the plain HTML code for the form:

## Text Fields

The name, email, and website fields are text input elements, and the comment field is a textarea. The HTML code looks like this:

Name: <input type="text" name="name">  
E-mail: <input type="text" name="email">  
Website: <input type="text" name="website">  
Comment: <textarea name="comment" rows="5" cols="40"></textarea>

## Radio Buttons

The gender fields are radio buttons and the HTML code looks like this:

Gender:  
<input type="radio" name="gender" value="female">Female  
<input type="radio" name="gender" value="male">Male

## The Form Element

The HTML code of the form looks like this:

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">

When the form is submitted, the form data is sent with method="post".

|  |  |
| --- | --- |
| Note | **What is the $\_SERVER["PHP\_SELF"] variable?** The $\_SERVER["PHP\_SELF"] is a super global variable that returns the filename of the currently executing script. |

So, the $\_SERVER["PHP\_SELF"] sends the submitted form data to the page itself, instead of jumping to a different page. This way, the user will get error messages on the same page as the form.

**What is the htmlspecialchars() function?**

The htmlspecialchars() function converts special characters to HTML entities. This means that it will replace HTML characters like < and > with &lt; and &gt;. This prevents attackers from exploiting the code by injecting HTML or Javascript code (Cross-site Scripting attacks) in forms.

We will also do two more things when the user submits the form:

1. Strip unnecessary characters (extra space, tab, newline) from the user input data (with the PHP trim() function)
2. Remove backslashes (\) from the user input data (with the PHP stripslashes() function)

We will name the function test\_input().

Now, we can check each $\_POST variable with the test\_input() function, and the script looks like this:

<?php  
// define variables and set to empty values  
$name = $email = $gender = $comment = $website = "";  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  $name = test\_input($\_POST["name"]);  
  $email = test\_input($\_POST["email"]);  
  $website = test\_input($\_POST["website"]);  
  $comment = test\_input($\_POST["comment"]);  
  $gender = test\_input($\_POST["gender"]);  
}  
function test\_input($data) {  
  $data = trim($data);  
  $data = stripslashes($data);  
  $data = htmlspecialchars($data);  
  return $data;  
}  
?>

In the following code we have added some new variables: $nameErr, $emailErr, $genderErr, and $websiteErr. These error variables will hold error messages for the required fields. We have also added an if else statement for each $\_POST variable. This checks if the $\_POST variable is empty (with the PHP empty() function). If it is empty, an error message is stored in the different error variables, and if it is not empty, it sends the user input data through the test\_input() function:

<?php  
// define variables and set to empty values  
$nameErr = $emailErr = $genderErr = $websiteErr = "";  
$name = $email = $gender = $comment = $website = "";  
if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  if (empty($\_POST["name"])) {  
    $nameErr = "Name is required";  
  } else {  
    $name = test\_input($\_POST["name"]);  
  }  
  if (empty($\_POST["email"])) {  
    $emailErr = "Email is required";  
  } else {  
    $email = test\_input($\_POST["email"]);  
  }  
  if (empty($\_POST["website"])) {  
    $website = "";  
  } else {  
    $website = test\_input($\_POST["website"]);  
  }  
  if (empty($\_POST["comment"])) {  
    $comment = "";  
  } else {  
    $comment = test\_input($\_POST["comment"]);  
  }  
  if (empty($\_POST["gender"])) {  
    $genderErr = "Gender is required";  
  } else {  
    $gender = test\_input($\_POST["gender"]);  
  }  
}  
?>

## PHP - Display The Error Messages

Then in the HTML form, we add a little script after each required field, which generates the correct error message if needed (that is if the user tries to submit the form without filling out the required fields):

<form method="post" action="<?php echo htmlspecialchars($\_SERVER["PHP\_SELF"]);?>">  
Name: <input type="text" name="name">  
<span class="error">\* <?php echo $nameErr;?></span>  
<br><br>  
E-mail:  
<input type="text" name="email">  
<span class="error">\* <?php echo $emailErr;?></span>  
<br><br>  
Website:  
<input type="text" name="website">  
<span class="error"><?php echo $websiteErr;?></span>  
<br><br>  
<label>Comment: <textarea name="comment" rows="5" cols="40"></textarea>  
<br><br>  
Gender:  
<input type="radio" name="gender" value="female">Female  
<input type="radio" name="gender" value="male">Male  
<span class="error">\* <?php echo $genderErr;?></span>  
<br><br>  
<input type="submit" name="submit" value="Submit">   
</form>

The next step is to validate the input data, that is "Does the Name field contain only letters and whitespace?", and "Does the E-mail field contain a valid e-mail address syntax?", and if filled out, "Does the Website field contain a valid URL?".

|  |  |
| --- | --- |
| Note | The preg\_match() function searches a string for pattern, returning true if the pattern exists, and false otherwise. |

<?php  
// define variables and set to empty values  
$nameErr = $emailErr = $genderErr = $websiteErr = "";  
$name = $email = $gender = $comment = $website = "";

if ($\_SERVER["REQUEST\_METHOD"] == "POST") {  
  if (empty($\_POST["name"])) {  
    $nameErr = "Name is required";  
  } else {  
    $name = test\_input($\_POST["name"]);  
    // check if name only contains letters and whitespace  
    if (!preg\_match("/^[a-zA-Z ]\*$/",$name)) {  
      $nameErr = "Only letters and white space allowed";   
    }  
  }  
  
  if (empty($\_POST["email"])) {  
    $emailErr = "Email is required";  
  } else {  
    $email = test\_input($\_POST["email"]);  
    // check if e-mail address is well-formed  
    if (!filter\_var($email, FILTER\_VALIDATE\_EMAIL)) {  
      $emailErr = "Invalid email format";   
    }  
  }  
  
  if (empty($\_POST["website"])) {  
    $website = "";  
  } else {  
    $website = test\_input($\_POST["website"]);  
    // check if URL address syntax is valid (this regular expression also allows dashes in the URL)  
    if (!preg\_match("/\b(?:(?:https?|ftp):\/\/|www\.)[-a-z0-9+&@#\/%?=~\_|!:,.;]\*[-a-z0-9+&@#\/%=~\_|]/i",$website)) {  
      $websiteErr = "Invalid URL";   
    }  
  }  
  
  if (empty($\_POST["comment"])) {  
    $comment = "";  
  } else {  
    $comment = test\_input($\_POST["comment"]);  
  }  
  
  if (empty($\_POST["gender"])) {  
    $genderErr = "Gender is required";  
  } else {  
    $gender = test\_input($\_POST["gender"]);  
  }  
}  
?>

## The PHP Date() Function

The PHP date() function formats a timestamp to a more readable date and time.

### Syntax

date(format,timestamp)

|  |  |
| --- | --- |
| Parameter | Description |
| format | Required. Specifies the format of the timestamp |
| timestamp | Optional. Specifies a timestamp. Default is the current date and time |

|  |  |
| --- | --- |
| Note | A timestamp is a sequence of characters, denoting the date and/or time at which a certain event occurred. |

## Get a Simple Date

The required *format* parameter of the date() function specifies how to format the date (or time).

Here are some characters that are commonly used for dates:

* d - Represents the day of the month (01 to 31)
* m - Represents a month (01 to 12)
* Y - Represents a year (in four digits)
* l (lowercase 'L') - Represents the day of the week

Other characters, like"/", ".", or "-" can also be inserted between the characters to add additional formatting.

The example below formats today's date in three different ways:

<?php  
echo "Today is " . date("Y/m/d") . "<br>";  
echo "Today is " . date("Y.m.d") . "<br>";  
echo "Today is " . date("Y-m-d") . "<br>";  
echo "Today is " . date("l");  
?>

## PHP Tip - Automatic Copyright Year

Use the date() function to automatically update the copyright year on your website:

&copy; 2010-<?php echo date("Y")?>

## Get a Simple Time

Here are some characters that are commonly used for times:

* h - 12-hour format of an hour with leading zeros (01 to 12)
* i - Minutes with leading zeros (00 to 59)
* s - Seconds with leading zeros (00 to 59)
* a - Lowercase Ante meridiem and Post meridiem (am or pm)

The example below outputs the current time in the specified format:

<?php  
echo "The time is " . date("h:i:sa");  
?>

# PHP - Include Files

The include (or require) statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

Including files is very useful when you want to include the same PHP, HTML, or text on multiple pages of a website.

## PHP include and require Statements

It is possible to insert the content of one PHP file into another PHP file (before the server executes it), with the include or require statement.

The include and require statements are identical, except upon failure:

* require will produce a fatal error (E\_COMPILE\_ERROR) and stop the script
* include will only produce a warning (E\_WARNING) and the script will continue

So, if you want the execution to go on and show users the output, even if the include file is missing, use the include statement. Otherwise, in case of FrameWork, CMS, or a complex PHP application coding, always use the require statement to include a key file to the flow of execution. This will help avoid compromising your application's security and integrity, just in-case one key file is accidentally missing.

Including files saves a lot of work. This means that you can create a standard header, footer, or menu file for all your web pages. Then, when the header needs to be updated, you can only update the header include file.

### Syntax

include 'filename';  
or  
require 'filename';

### Example 1

Assume we have a standard footer file called "footer.php", that looks like this:

<?php  
echo "<p>Copyright &copy; 1999-" . date("Y") . " W3Schools.com</p>";  
?>

To include the footer file in a page, use the include statement:

<html>  
<body>  
  
<h1>Welcome to my home page!</h1>  
<p>Some text.</p>  
<p>Some more text.</p>  
<?php include 'footer.php';?>  
</body>  
</html>

### Example 2

Assume we have a file called "vars.php", with some variables defined:

<?php  
$color='red';  
$car='BMW';  
?>

Then, if we include the "vars.php" file, the variables can be used in the calling file:

## Example

<html>  
<body>  
<h1>Welcome to my home page!</h1>  
<?php include 'vars.php';  
echo "I have a $color $car.";  
?>  
</body>  
</html>

# Chapter Three: Files and Directories

File handling is an important part of any web application. You often need to open and process a file for different tasks.

## PHP Manipulating Files

PHP has several functions for creating, reading, uploading, and editing files.

|  |  |
| --- | --- |
| Note | **Be careful when manipulating files!** When you are manipulating files you must be very careful. You can do a lot of damage if you do something wrong. Common errors are: editing the wrong file, filling a hard-drive with garbage data, and deleting the content of a file by accident. |

## PHP readfile() Function

The readfile() function reads a file and writes it to the output buffer.

Assume we have a text file called "webdictionary.txt", stored on the server, that looks like this:

AJAX = Asynchronous JavaScript and XML  
CSS = Cascading Style Sheets  
HTML = Hyper Text Markup Language  
PHP = PHP Hypertext Preprocessor  
SQL = Structured Query Language  
SVG = Scalable Vector Graphics  
XML = EXtensible Markup Language

## Example

<?php  
echo readfile("webdictionary.txt");  
?>

The readfile() function is useful if all you want to do is open up a file and read its contents.

## PHP Open File - fopen()

A better method to open files is with the fopen() function. This function gives you more options than the readfile() function.

The first parameter of fopen() contains the name of the file to be opened and the second parameter specifies in which mode the file should be opened. The following example also generates a message if the fopen() function is unable to open the specified file:

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
echo fread($myfile,filesize("webdictionary.txt"));  
fclose($myfile);  
?>

**Tip:** The fread() and the fclose() functions will be explained below.

The file may be opened in one of the following modes:

|  |  |
| --- | --- |
| Modes | Description |
| r | **Open a file for read only**. File pointer starts at the beginning of the file |
| w | **Open a file for write only**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file |
| a | **Open a file for write only**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist |
| x | **Creates a new file for write only**. Returns FALSE and an error if file already exists |
| r+ | **Open a file for read/write**. File pointer starts at the beginning of the file |
| w+ | **Open a file for read/write**. Erases the contents of the file or creates a new file if it doesn't exist. File pointer starts at the beginning of the file |
| a+ | **Open a file for read/write**. The existing data in file is preserved. File pointer starts at the end of the file. Creates a new file if the file doesn't exist |
| x+ | **Creates a new file for read/write**. Returns FALSE and an error if file already exists |

## PHP Read File - fread()

The fread() function reads from an open file.

The first parameter of fread() contains the name of the file to read from and the second parameter specifies the maximum number of bytes to read.

The following PHP code reads the "webdictionary.txt" file to the end:

fread($myfile,filesize("webdictionary.txt"));

## PHP Close File - fclose()

The fclose() function is used to close an open file.

|  |  |
| --- | --- |
| Note | It's a good programming practice to close all files after you have finished with them. You don't want an open file running around on your server taking up resources! |

The fclose() requires the name of the file (or a variable that holds the filename) we want to close:

<?php  
$myfile = fopen("webdictionary.txt", "r");  
// some code to be executed....  
fclose($myfile);  
?>

## PHP Read Single Line - fgets()

The fgets() function is used to read a single line from a file.

The example below outputs the first line of the "webdictionary.txt" file:

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
echo fgets($myfile);  
fclose($myfile);  
?>

**Note:** After a call to the fgets() function, the file pointer has moved to the next line.

## PHP Check End-Of-File - feof()

The feof() function checks if the "end-of-file" (EOF) has been reached.

The feof() function is useful for looping through data of unknown length.

The example below reads the "webdictionary.txt" file line by line, until end-of-file is reached:

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
// Output one line until end-of-file  
while(!feof($myfile)) {  
  echo fgets($myfile) . "<br>";  
}  
fclose($myfile);  
?>

## PHP Read Single Character - fgetc()

The fgetc() function is used to read a single character from a file.The example below reads the "webdictionary.txt" file character by character, until end-of-file is reached:

<?php  
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");  
// Output one character until end-of-file  
while(!feof($myfile)) {  
  echo fgetc($myfile);  
}  
fclose($myfile);  
?>

**Note:** After a call to the fgetc() function, the file pointer moves to the next character.

## PHP Create File - fopen()

The fopen() function is also used to create a file. Maybe a little confusing, but in PHP, a file is created using the same function used to open files.

If you use fopen() on a file that does not exist, it will create it, given that the file is opened for writing (w) or appending (a).

The example below creates a new file called "testfile.txt". The file will be created in the same directory where the PHP code resides:

$myfile = fopen("testfile.txt", "w") ;

## PHP File Permissions

If you are having errors when trying to get this code to run, check that you have granted your PHP file access to write information to the hard drive.

## PHP Write to File - fwrite()

The fwrite() function is used to write to a file.

The first parameter of fwrite() contains the name of the file to write to and the second parameter is the string to be written.

The example below writes a couple of names into a new file called "newfile.txt":

<?php  
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");  
$txt = "John Doe\n";  
fwrite($myfile, $txt);  
$txt = "Jane Doe\n";  
fwrite($myfile, $txt);  
fclose($myfile);  
?>

Notice that we wrote to the file "newfile.txt" twice. Each time we wrote to the file we sent the string $txt that first contained "John Doe" and second contained "Jane Doe". After we finished writing, we closed the file using the fclose() function.

## PHP Overwriting

Now that "newfile.txt" contains some data we can show what happens when we open an existing file for writing. All the existing data will be ERASED and we start with an empty file.

In the example below we open our existing file "newfile.txt", and write some new data into it:

<?php  
$myfile = fopen("newfile.txt", "w") or die("Unable to open file!");  
$txt = "Mickey Mouse\n";  
fwrite($myfile, $txt);  
$txt = "Minnie Mouse\n";  
fwrite($myfile, $txt);  
fclose($myfile);  
?>

If we now open the "newfile.txt" file, both John and Jane have vanished, and only the data we just wrote will be present.

PHP - File Upload

With PHP, it is easy to upload files to the server.

However, with ease comes danger, so always be careful when allowing file uploads!

## Configure The "php.ini" File

First, ensure that PHP is configured to allow file uploads.

In your "php.ini" file, search for the file\_uploads directive, and set it to On:

file\_uploads = On

Next, create an HTML form that allow users to choose the image file they want to upload:

<!DOCTYPE html>  
<html>  
<body>  
<form action="upload.php" method="post" enctype="multipart/form-data">  
    Select image to upload:  
    <input type="file" name="fileToUpload" id="fileToUpload">  
    <input type="submit" value="Upload Image" name="submit">  
</form>  
</body>  
</html>

Some rules to follow for the HTML form above:

* Make sure that the form uses method="post"
* The form also needs the following attribute: enctype="multipart/form-data". It specifies which content-type to use when submitting the form

Without the requirements above, the file upload will not work.

Other things to notice:

* The type="file" attribute of the <input> tag shows the input field as a file-select control, with a "Browse" button next to the input control

The form above sends data to a file called "upload.php", which we will create next.

## Create The Upload File PHP Script

The "upload.php" file contains the code for uploading a file:

<?php  
$target\_dir = "uploads/";  
$target\_file = $target\_dir . basename($\_FILES["fileToUpload"]["name"]);  
$uploadOk = 1;  
$imageFileType = pathinfo($target\_file,PATHINFO\_EXTENSION);  
// Check if image file is a actual image or fake image  
if(isset($\_POST["submit"])) {  
    $check = getimagesize($\_FILES["fileToUpload"]["tmp\_name"]);  
    if($check !== false) {  
        echo "File is an image - " . $check["mime"] . ".";  
        $uploadOk = 1;  
    } else {  
        echo "File is not an image.";  
        $uploadOk = 0;  
    }  
}  
?>

PHP script explained:

* $target\_dir = "uploads/" - specifies the directory where the file is going to be placed
* $target\_file specifies the path of the file to be uploaded
* $uploadOk=1 is not used yet (will be used later)
* $imageFileType holds the file extension of the file
* Next, check if the image file is an actual image or a fake image

|  |  |
| --- | --- |
| Note | **Note:** You will need to create a new directory called "uploads" in the directory where "upload.php" file resides. The uploaded files will be saved there. |

## Check if File Already Exists

Now we can add some restrictions.

First, we will check if the file already exists in the "uploads" folder. If it does, an error message is displayed, and $uploadOk is set to 0:

// Check if file already exists  
if (file\_exists($target\_file)) {  
    echo "Sorry, file already exists.";  
    $uploadOk = 0;  
}

## Limit File Size

The file input field in our HTML form above is named "fileToUpload".

Now, we want to check the size of the file. If the file is larger than 500kb, an error message is displayed, and $uploadOk is set to 0:

 // Check file size  
if ($\_FILES["fileToUpload"]["size"] > 500000) {  
    echo "Sorry, your file is too large.";  
    $uploadOk = 0;  
}

## Limit File Type

The code below only allows users to upload JPG, JPEG, PNG, and GIF files. All other file types gives an error message before setting $uploadOk to 0:

// Allow certain file formats  
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"  
&& $imageFileType != "gif" ) {  
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";  
    $uploadOk = 0;  
}

The complete "upload.php" file now looks like this:

<?php  
$target\_dir = "uploads/";  
$target\_file = $target\_dir . basename($\_FILES["fileToUpload"]["name"]);  
$uploadOk = 1;  
$imageFileType = pathinfo($target\_file,PATHINFO\_EXTENSION);  
// Check if image file is a actual image or fake image  
if(isset($\_POST["submit"])) {  
    $check = getimagesize($\_FILES["fileToUpload"]["tmp\_name"]);  
    if($check !== false) {  
        echo "File is an image - " . $check["mime"] . ".";  
        $uploadOk = 1;  
    } else {  
        echo "File is not an image.";  
        $uploadOk = 0;  
    }  
}  
// Check if file already exists  
if (file\_exists($target\_file)) {  
    echo "Sorry, file already exists.";  
    $uploadOk = 0;  
}  
// Check file size  
if ($\_FILES["fileToUpload"]["size"] > 500000) {  
    echo "Sorry, your file is too large.";  
    $uploadOk = 0;  
}  
// Allow certain file formats  
if($imageFileType != "jpg" && $imageFileType != "png" && $imageFileType != "jpeg"  
&& $imageFileType != "gif" ) {  
    echo "Sorry, only JPG, JPEG, PNG & GIF files are allowed.";  
    $uploadOk = 0;  
}  
// Check if $uploadOk is set to 0 by an error  
if ($uploadOk == 0) {  
    echo "Sorry, your file was not uploaded.";  
// if everything is ok, try to upload file  
} else {  
    if (move\_uploaded\_file($\_FILES["fileToUpload"]["tmp\_name"], $target\_file)) {  
        echo "The file ". basename( $\_FILES["fileToUpload"]["name"]). " has been uploaded.";  
    } else {  
        echo "Sorry, there was an error uploading your file.";  
    }  
}  
?>

# Chapter 4 - Database Driven Website with PHP and MySQL

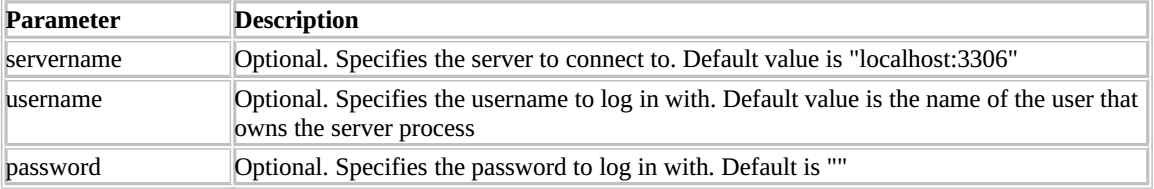
## What is MySQL?

MySQL is the most popular open-source database system. MySQL is a database. The data in MySQL is stored in database objects called tables. A table is a collections of related data entries and it consists of columns and rows. Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".

**PHP MySQL Connect to a Database**

The free MySQL database is very often used with PHP. Before you can access data in a database, you must create a connection to the database. In PHP, this is done with the mysql\_connect() function. Syntax

mysql\_connect(servername,username,password);

**Note**: There are more available parameters, but the ones listed above are the most important.

Example

In the following example we store the connection in a variable ($con) for later use in the script. The "die" part will be executed if the connection fails:

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

// some code

?>

Closing a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the mysql\_close() function:

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

// some code

mysql\_close($con);

?>

**Create Database**

The CREATE DATABASE statement is used to create a database in MySQL.Synax

CREATE DATABASE database\_name;

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

Example - creates a database called "my\_db":

<?php

$con = mysql\_connect("localhost","root","password");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

if (mysql\_query("CREATE DATABASE my\_db",$con))

{

echo "Database created";

}

else

{

echo "Error creating database: " . mysql\_error();

}

mysql\_close($con);

?>

**Create a Table**

The CREATE TABLE statement is used to create a table in MySQL. Syntax

CREATE TABLE table\_name  
(  
column\_name1 data\_type,  
column\_name2 data\_type,  
column\_name3 data\_type,  
....  
);

We must add the CREATE TABLE statement to the mysql\_query() function to execute the command. E.g.

The following example creates a table named "Persons", with three columns. The column names will be

"FirstName", "LastName" and "Age":

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

$sql = "CREATE TABLE Persons(FirstName varchar(15), LastName varchar(15),Age int )";

mysql\_query($sql,$con);

mysql\_close($con);

?>

**Important**: A database must be selected before a table can be created. The database is selected with the mysql\_select\_db() function.

**Note**: When you create a database field of type varchar, you must specify the maximum length of the field, e.g. varchar(15).

The data type specifies what type of data the column can hold.

To see the table column names and data types use in mysql command prompt: **show fields from tableName;**

**Primary Keys and Auto Increment Fields**  
Each table should have a primary key field.  
A primary key is used to uniquely identify the rows in a table. Each primary key value must be unique within the table. Furthermore, the primary key field cannot be null because the database engine requires a value to locate the record.  
The following example sets the personID field as the primary key field. The primary key field is often an ID  
number, and is often used with the AUTO\_INCREMENT setting. AUTO\_INCREMENT automatically  
increases the value of the field by 1 each time a new record is added. To ensure that the primary key field  
cannot be null, we must add the NOT NULL setting to the field. Example

$sql = "CREATE TABLE Persons

(

personID int NOT NULL AUTO\_INCREMENT,

PRIMARY KEY(personID),

FirstName varchar(15),

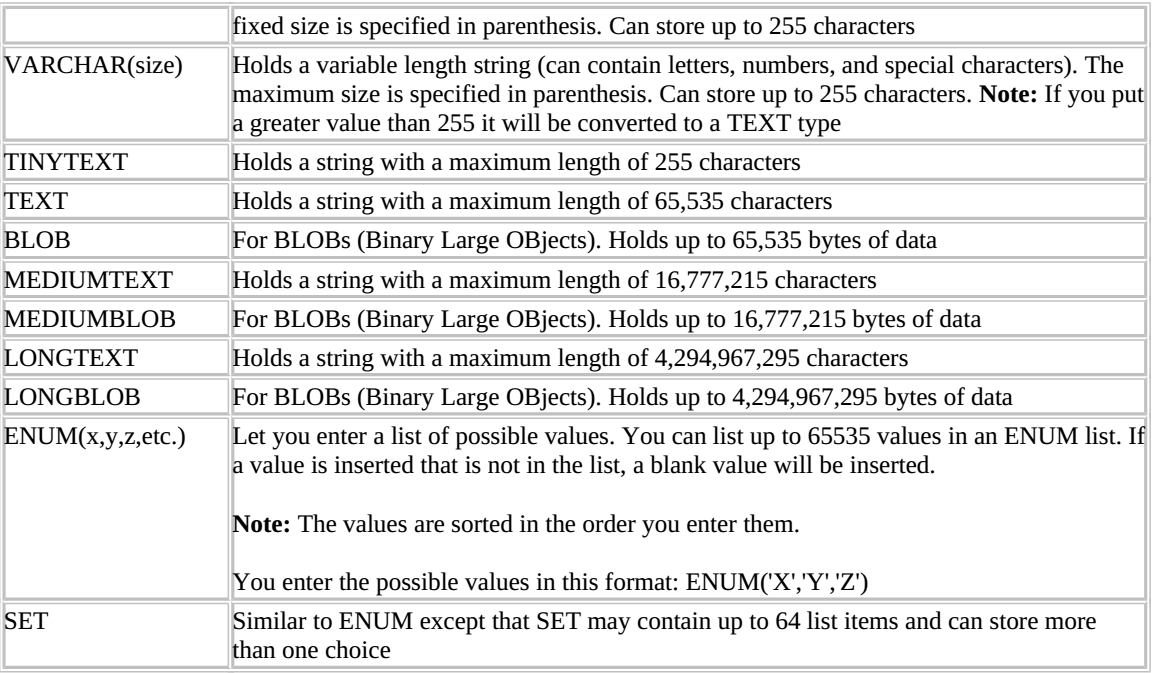
LastName varchar(15),

Age int

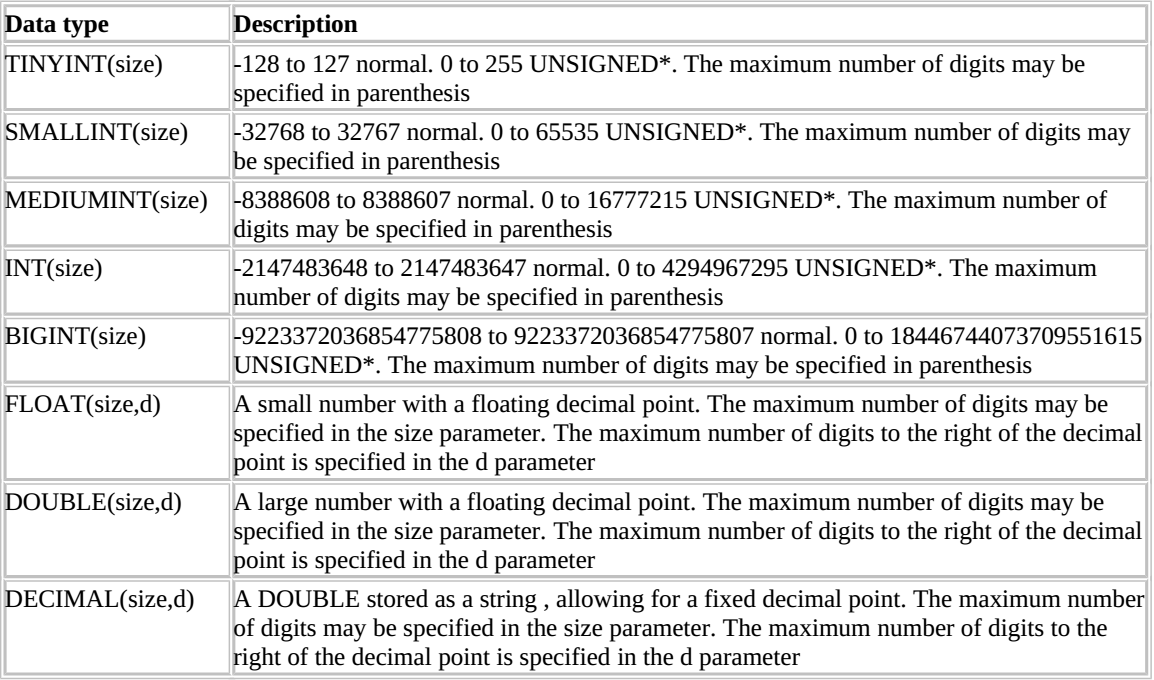
)";

mysql\_query($sql,$con);

**MySQL Data Types-** text types

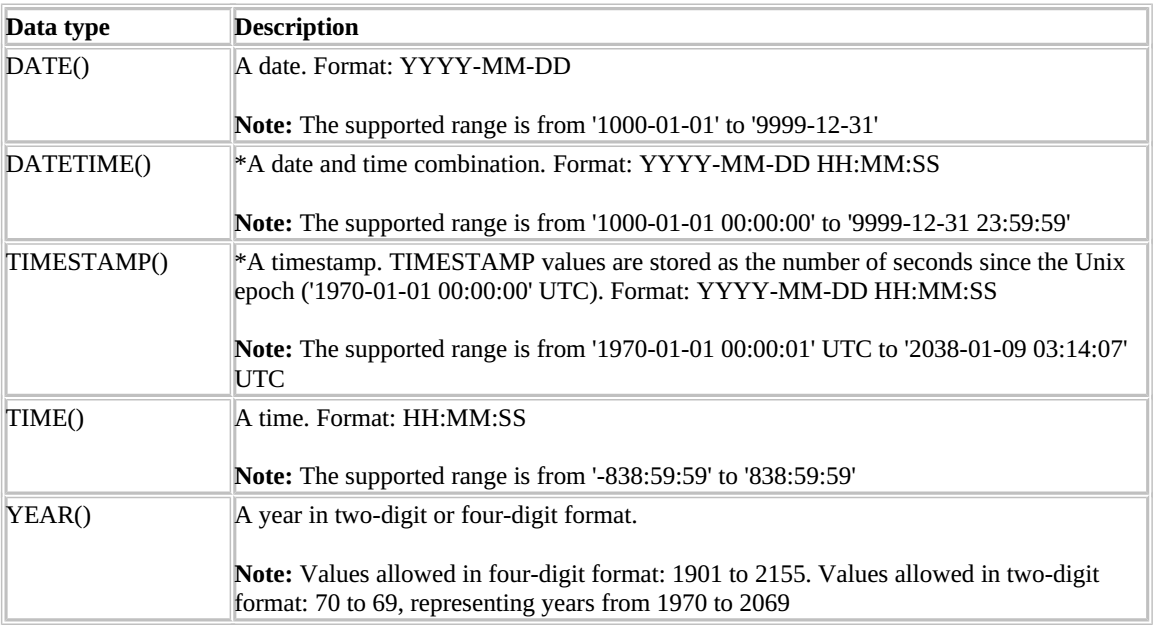


**Number types:**



\*The integer types have an extra option called UNSIGNED. Normally, the integer goes from an negative to  
positive value. Adding the UNSIGNED attribute will move that range up so it starts at zero instead of a negative  
number.

**Date types:**



\*Even if DATETIME and TIMESTAMP return the same format, they work very differently. In an INSERT or UPDATE query, the TIMESTAMP automatically set itself to the current date and time. TIMESTAMP also  
accepts various formats, like YYYYMMDDHHMMSS, YYMMDDHHMMSS, YYYYMMDD, or YYMMDD.

**Queries**

A query is a question or a request. With MySQL, we can query a database for specific information and have a record set returned.

Look at the following query:

SELECT LastName FROM Persons;

The query above selects all the data in the "LastName" column from the "Persons" table, and will return a recordset like this:



**Facts about MySQL Database**

One great thing about MySQL is that it can be scaled down to support embedded database applications. Perhaps it is because of this reputation that many people believe that MySQL can only handle small to medium-sized systems.

The truth is that MySQL is the de-facto standard database for web sites that support huge volumes of both data and end users (like Friendster, Yahoo, and Google).

**Insert Data Into a Database Table**

The INSERT INTO statement is used to add new records to a database table. Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

INSERT INTO table\_name VALUES (value1, value2, value3,...);

The second form specifies both the column names and the values to be inserted:

INSERT INTO table\_name (column1, column2, column3,...) VALUES (value1, value2, value3,...);

To get PHP to execute the statements above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

Example

In the previous chapter we created a table named "Persons", with three columns; "Firstname", "Lastname" and "Age". We will use the same table in this example. The following example adds two new records to the "Persons" table:

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

mysql\_query("INSERT INTO Persons (FirstName, LastName, Age) VALUES ('Peter', 'Griffin', '35')");

mysql\_query("INSERT INTO Persons (FirstName, LastName, Age) VALUES ('Glenn', 'Quagmire', '33')");

mysql\_close($con);

?>

**Insert Data From a Form Into a Database**

**Now we will create an HTML form that can be used to add new records to the "Persons" table. Here is the HTML form:**

**<html>**

**<body>**

**<form action="sql5.php" method="post">**

**Firstname: <input type="text" name="firstname" />**

**Lastname: <input type="text" name="lastname" />**

**Age: <input type="text" name="age" />**

**<input type="submit" />**

**</form>**

**</body>**

**</html>**

**When a user clicks the submit button in the HTML form in the example above, the form data is sent to "insert.php".**

**The "insert.php" file connects to a database, and retrieves the values from the form with the PHP $\_POST variables.**

**Then, the mysql\_query() function executes the INSERT INTO statement, and a new record will be added to the "Persons" table. Here is the "insert.php" page:**

**<?php**

**$con = mysql\_connect("localhost","root","K236k236");**

**if (!$con)**

**{**

**die('Could not connect: ' . mysql\_error());**

**}**

**mysql\_select\_db("my\_db", $con);**

**$sql="INSERT INTO Persons (FirstName, LastName, Age) VALUES('$\_POST[firstname]','$\_POST[lastname]','$\_POST[age]')";**

**if (!mysql\_query($sql,$con))**

**{**

**die('Error: ' . mysql\_error());**

**}**

**echo "1 record added";**

**mysql\_close($con)**

**?>**

**Select Data From a Database Table**

The SELECT statement is used to select data from a database. Syntax

SELECT column\_name(s) FROM table\_name;

To learn more about SQL, please visit our SQL tutorial. To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all the data stored in the "Persons" table (The \* character selects all the data in the table):

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

$result = mysql\_query("SELECT \* FROM Persons");

while($row = mysql\_fetch\_array($result))

{

echo $row['FirstName'] . " " . $row['LastName'];

echo "<br />";

}

mysql\_close($con);

?>

The example above stores the data returned by the mysql\_query() function in the $result variable.

Next, we use the mysql\_fetch\_array() function to return the first row from the recordset as an array. Each call to

mysql\_fetch\_array() returns the next row in the recordset. The while loop loops through all the records in the

recordset. To print the value of each row, we use the PHP $row variable ($row['FirstName'] and $row['LastName']).

The output of the code above will be:

Alex Alemayehu

Almaz Ayana

**Display the Result in an HTML Table**

The following example selects the same data as the example above, but will display the data in an HTML table:

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

$result = mysql\_query("SELECT \* FROM Persons");

echo "<table border='1'>

<tr>

<th>Firstname</th>

<th>Lastname</th>

</tr>";

while($row = mysql\_fetch\_array($result))

{

echo "<tr>";

echo "<td>" . $row['FirstName'] . "</td>";

echo "<td>" . $row['LastName'] . "</td>";

echo "</tr>";

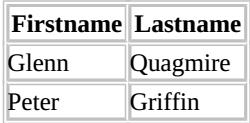
}

echo "</table>";

mysql\_close($con);

?>

The output of the code above will be:



**PHP MySQL The Where Clause**

The WHERE clause is used to filter records.

The WHERE clause is used to extract only those records that fulfill a specified criterion. Syntax

SELECT column\_name(s) FROM table\_name WHERE column\_name operator value;

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all rows from the "Persons" table where "FirstName='Peter':

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

$result = mysql\_query("SELECT \* FROM Persons WHERE FirstName='Peter'");

while($row = mysql\_fetch\_array($result))

{

echo $row['FirstName'] . " " . $row['LastName'];

echo "<br />";

}

?>

The output of the code above will be:

Peter Griffin

**PHP MySQL Order By Keyword**

The ORDER BY keyword is used to sort the data in a recordset.

The ORDER BY keyword sort the records in ascending order by default. If you want to sort the records in a descending order, you can use the DESC keyword. Syntax

SELECT column\_name(s) FROM table\_name ORDER BY column\_name(s) ASC|DESC;

Example

The following example selects all the data stored in the "Persons" table, and sorts the result by the "Age" column:

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

$result = mysql\_query("SELECT \* FROM Persons ORDER BY age");

while($row = mysql\_fetch\_array($result))

{

echo $row['FirstName'];

echo " " . $row['LastName'];

echo " " . $row['Age'];

echo "<br />";

}

mysql\_close($con);

?>

The output of the code above will be:

Glenn Quagmire 33

Peter Griffin 35

John Henry 56

**Order by Two Columns**

It is also possible to order by more than one column. When ordering by more than one column, the second column is only used if the values in the first column are equal:

SELECT column\_name(s) FROM table\_name ORDER BY column1, column2

**PHP MySQL Update**

The UPDATE statement is used to modify data in a table. The UPDATE statement is used to update existing records in a table. Syntax

UPDATE table\_name SET column1=value, column2=value2,...WHERE some\_column=some\_value;

**Note**: Notice the WHERE clause in the UPDATE syntax. The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

Example

Earlier in the tutorial we created a table named "Persons". Here is how it looks:

FirstName LastName Age

Peter Griffin 35

Glenn Quagmire 33

John Henry 56

The following example updates some data in the "Persons" table:

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con)

{

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

mysql\_query("UPDATE Persons SET Age = '36' WHERE FirstName = 'Peter' AND LastName = 'Griffin'");

mysql\_close($con);

?>

After the update, the "Persons" table will look like this:

FirstName LastName Age

Peter Griffin 36

Glenn Quagmire 33

John Henry 56

**PHP MySQL Delete**

The DELETE FROM statement is used to delete records from a database table. Syntax

DELETE FROM table\_name WHERE some\_column = some\_value;

**Note**: Notice the WHERE clause in the DELETE syntax. The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

To get PHP to execute the statement above we must use the mysql\_query() function. This function is used to send a query or command to a MySQL connection.

Example

Look at the following "Persons" table:



The following example deletes all the records in the "Persons" table where LastName='Griffin':

<?php

$con = mysql\_connect("localhost","root","K236k236");

if (!$con){

die('Could not connect: ' . mysql\_error());

}

mysql\_select\_db("my\_db", $con);

mysql\_query("DELETE FROM Persons WHERE LastName='Griffin'");

mysql\_close($con);

?>

After the deletion, the table will look like this:



**PHP Database ODBC**

ODBC is an Application Programming Interface (API) that allows you to connect to a data source (e.g. an MS Access database).

**Create an ODBC Connection**

With an ODBC connection, you can connect to any database, on any computer in your network, as long as an ODBC connection is available.

Here is how to create an ODBC connection to a MS Access Database:

1. Open the Administrative Tools icon in your Control Panel.

2. Double-click on the Data Sources (ODBC) icon inside.

3. Choose the System DSN tab.

4. Click on Add in the System DSN tab.

5. Select the Microsoft Access Driver. Click Finish.

6. In the next screen, click Select to locate the database.

7. Give the database a Data Source Name (DSN).

8. Click OK.

**Note** that this configuration has to be done on the computer where your web site is located. If you are running Internet Information Server (IIS) on your own computer, the instructions above will work, but if your web site is located on a remote server, you have to have physical access to that server, or ask your web host to set up a DSN for you to use.

**Connecting to an ODBC**

The odbc\_connect() function is used to connect to an ODBC data source. The function takes four parameters: the data source name, username, password, and an optional cursor type. The odbc\_exec() function is used to execute an SQL statement.

Example

The following example creates a connection to a DSN called northwind, with no username and no password. It then creates an SQL and executes it:

$conn=odbc\_connect('northwind','','');

$sql="SELECT \* FROM customers";

$rs=odbc\_exec($conn,$sql);

**Retrieving Records**

The odbc\_fetch\_row() function is used to return records from the result-set. This function returns true if it is able to return rows, otherwise false. The function takes two parameters: the ODBC result identifier and an optional row number:

odbc\_fetch\_row($rs)

**Retrieving Fields from a Record**

The odbc\_result() function is used to read fields from a record. This function takes two parameters: the ODBC result identifier and a field number or name. The code line below returns the value of the first field from the record:

$compname=odbc\_result($rs,1);

The code line below returns the value of a field called "CompanyName":

$compname=odbc\_result($rs,"CompanyName");

**Closing an ODBC Connection**

The odbc\_close() function is used to close an ODBC connection.

odbc\_close($conn);

An ODBC Example

The following example shows how to first create a database connection, then a result-set, and then display the data in an HTML table.

<html>

<body>

<?php

$conn=odbc\_connect('northwind','','');

if (!$conn)

{exit("Connection Failed: " . $conn);}

$sql="SELECT \* FROM customers";

$rs=odbc\_exec($conn,$sql);

if (!$rs)

{exit("Error in SQL");}

echo "<table><tr>";

echo "<th>Companyname</th>";

echo "<th>Contactname</th></tr>";

while (odbc\_fetch\_row($rs))

{

$compname=odbc\_result($rs,"CompanyName");

$conname=odbc\_result($rs,"ContactName");

echo "<tr><td>$compname</td>";

echo "<td>$conname</td></tr>";

}

odbc\_close($conn);

echo "</table>";

?>

</body>

</html>

# Chapter 5- Session and Cookies

HTTP is a stateless protocol. This means that the protocol has no built-in way of maintaining state between two transactions. When a user requests one page, followed by another, HTTP does not provide a way for you to tell that both requests came from the same user.  
The idea of session control is to be able to track a user during a single session on a website. If you can do this, you can easily support logging in a user and showing content according to her authorization level or personal preferences.

As web applications have matured, the need for *statefulness* has become a common requirement. Stateful web applications, meaning applications that keep track of a particular visitor’s information as he travels throughout a site.

**Using Cookies**  
Cookies are a way for a server to store information on the user’s machine. This is one way that a site can remember or track a user over the course of a visit. A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

**Create Cookies with PHP**

A cookie is created with the setcookie() function. Syntax

setcookie(name, value, expire, path, domain, secure, httponly);

Only the name parameter is required. All other parameters are optional.

**PHP Create/Retrieve a Cookie**

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 \* 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $\_COOKIE). We also use the isset() function to find out if the cookie is set: Example

<?php  
$cookie\_name = "user";  
$cookie\_value = "John Doe";  
setcookie($cookie\_name, $cookie\_value, time() + (86400 \* 30), "/"); // 86400 = 1 day  
?>  
<html>  
<body>  
<?php  
if(!isset($\_COOKIE[$cookie\_name])) {  
    echo "Cookie named '" . $cookie\_name . "' is not set!";  
} else {  
    echo "Cookie '" . $cookie\_name . "' is set!<br>";  
    echo "Value is: " . $\_COOKIE[$cookie\_name];  
}  
?>  
</body>  
</html>

**Modify a Cookie Value**

To modify a cookie, just set (again) the cookie using the setcookie() function: Example

<?php  
$cookie\_name = "user";  
$cookie\_value = "Alex Porter";  
setcookie($cookie\_name, $cookie\_value, time() + (86400 \* 30), "/");  
?>  
<html>  
<body>  
<?php  
if(!isset($\_COOKIE[$cookie\_name])) {  
    echo "Cookie named '" . $cookie\_name . "' is not set!";  
} else {  
    echo "Cookie '" . $cookie\_name . "' is set!<br>";  
    echo "Value is: " . $\_COOKIE[$cookie\_name];  
}  
?>  
</body>  
</html>

**Delete a Cookie**

To delete a cookie, use the setcookie() function with an expiration date in the past: Example

<?php  
// set the expiration date to one hour ago  
setcookie("user", "", time() - 3600);  
?>  
<html>  
<body>  
<?php  
echo "Cookie 'user' is deleted.";  
?>  
</body>  
</html>

**Check if Cookies are enabled**

The following example creates a small script that checks whether cookies are enabled. First, try to create a test cookie with the setcookie() function, then count the $\_COOKIE array variable: Example

<?php  
setcookie("test\_cookie", "test", time() + 3600, '/');  
if(count($\_COOKIE) > 0) {  
    echo "Cookies are enabled.";  
} else {  
    echo "Cookies are disabled.";  
}  
?>

## What is a PHP Session?

A session is a way to store information (in variables) to be used across multiple pages. Unlike a cookie, the information is not stored on the user’s computer.

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

The basic steps to use session are:

* starting a session
* registering a session variable
* using the session variable
* deregistering the variable and destroying the session

## Start a PHP Session

A session is started with the session\_start() function. Session variables are set with the PHP global variable: $\_SESSION. Now, let's create a new page called "demo\_session1.php". In this page, we start a new PHP session and set some session variables: Example

<?php  
session\_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
<?php  
// Set session variables  
$\_SESSION["favcolor"] = "green";  
$\_SESSION["favanimal"] = "cat";  
echo "Session variables are set.";  
?>  
</body>  
</html>

|  |  |
| --- | --- |
| **Note** | **Note:** The session\_start() function must be the very first thing in your document. Before any HTML tags. |

## Get PHP Session Variable Values

Next, we create another page called "demo\_session2.php". From this page, we will access the session information we set on the first page ("demo\_session1.php").

Notice that session variables are not passed individually to each new page, instead they are retrieved from the session we open at the beginning of each page (session\_start()).

Also notice that all session variable values are stored in the global $\_SESSION variable: Example

<?php  
session\_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
<?php  
// Echo session variables that were set on previous page  
echo "Favorite color is " . $\_SESSION["favcolor"] . ".<br>";  
echo "Favorite animal is " . $\_SESSION["favanimal"] . ".";  
?>  
</body>  
</html>

Another way to show all the session variable values for a user session is to run the following code: Example

<?php  
session\_start();  
?>  
<!DOCTYPE html>  
<html>  
<body>  
<?php  
print\_r($\_SESSION);  
?>  
</body>  
</html>

**Modify a PHP Session Variable**

To change a session variable, just overwrite it: Example

<?php  
session\_start();  
$\_SESSION["favcolor"] = "yellow";  
print\_r($\_SESSION);  
?>

**Destroy a PHP Session**

To remove all global session variables and destroy the session, use session\_unset() and session\_destroy(): e.g.

<?php  
session\_start();  
session\_unset();   
  
// destroy the session   
session\_destroy();   
?>

Sessions vs. Cookies  
Obviously, both are easy to use in PHP, but the true question is when to use one or the other.  
Sessions have the following advantages over cookies:

* They are generally more secure (because the data is being retained on the server).
* They allow for more data to be stored.
* They can be used without cookies.

Whereas cookies have the following advantages over sessions:

* They are easier to program.
* They require less of the server.
* They can be made to last far longer.

In general, to store and retrieve just a couple of small pieces of information, or to store information  
for a longer duration, use cookies. For most of your Web applications, though, you’ll use sessions.

## More example on Session and Cookies with database.

Question:

* create a login script from a stored database
* use session and cookies for authentication and autorizattion

1. Create a database called login

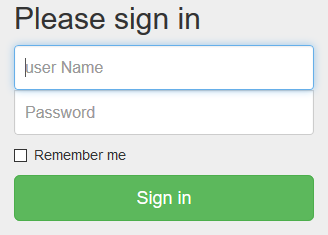
2. Create a table called users

3. And populate you table with some data.

4. Create a login form

5. Connect with you database, write Sql and communicate with your form by running your SQL.

6. Create Session and Cookie variables to authenticate use while navigating from one page to another.



* [CREATE](http://localhost/phpmyadmin/url.php?url=http%3A%2F%2Fdev.mysql.com%2Fdoc%2Frefman%2F5.5%2Fen%2Fcreate-database.html&server=0&token=69dec9ac611517ca5a15abb8f368cab5) [DATABASE](http://localhost/phpmyadmin/url.php?url=http%3A%2F%2Fdev.mysql.com%2Fdoc%2Frefman%2F5.5%2Fen%2Fcreate-database.html&server=0&token=69dec9ac611517ca5a15abb8f368cab5) LOGIN;
* CREATE TABLE IF NOT EXISTS `users` ( `UserName` varchar(255) NOT NULL, `Password` varchar(255) NOT NULL, `FirstName` varchar(255) NOT NULL, `LastName` varchar(255) NOT NULL, `Department` enum('IT','IS','CS','SW') NOT NULL,`age` int(11) NOT NULL) ENGINE=InnoDB;
* INSERT INTO `users` (`UserName`, `Password`, `FirstName`, `LastName`, `Department`, `age`) VALUES('abe', '123abc', 'kebede', 'mekonen', 'IS', 26),('alex', '123', 'aleex', 'abrham', 'SW', 24);

## Creating our form

<form class="form-signin" method="post">

<h2>Please sign in</h2>

Username <input type="text" name="txtuname" placeholder="user Name" required autofocus>

Password <input type="password" name="txtpwd" placeholder="Password" required>

<input type="checkbox" value="ON" name="chkbtn"> Remember me

<button class="btn btn-lg btn-success btn-block" type="submit" name="btnlogin">Sign in</button>

</form>

## Creating database connection

<?php

$con=new mysqli("localhost","root","","login");

?>

Save this file as db\_con.php so that we can include to every file which need a database connection.

## Writing the login script

<?php

if(isset($\_POST["btnlogin"])){

$uname=$\_POST["txtuname"];

$pwd=$\_POST["txtpwd"];

if(isset($\_POST["chkbtn"])) {

$remeber=$\_POST["chkbtn"];

}else{

$remeber="OFF";

}

require 'db\_con.php';

$qry="SELECT \* FROM `users` WHERE `UserName`='$uname' AND `Password`='$pwd'";

$run=$con->query($qry);

$number=$run->num\_rows;

if($number>0){

$\_SESSION["UserName"]=$uname;

if($remeber=="ON"){

$cookie\_name="UName";

setcookie($cookie\_name, $uname, time() + (60), "/");

}

header('location:list.php');

}else{

echo 'Incorect UserName or Password,try again';

}

}

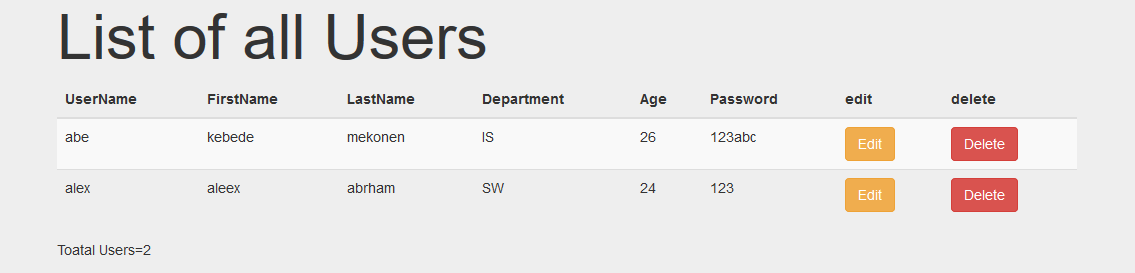
?>

## Check if user logged correctly

as we see in the above script the user will go automatically to **list.php** after login.

so we have to check if logged correctly or not at list php.

think list.php look like this.



<?php

session\_start();

if(isset($\_SESSION["UserName"])){

$loogeduser=$\_SESSION["UserName"];

}else{

header('location:index.php');

}

?>

<!DOCTYPE html>

<html lang="en">

<h1>List of all Users</h1>

<table>

<thead>

<tr>

<th>UserName</th> <th>FirstName</th> <th>LastName</th>

<th>Department</th> <th>Age</th> <th>Password</th>

<th>edit</th> <th>delete</th>

</tr>

</thead>

<tbody>

<?php

require 'db\_con.php';

$qry="SELECT \* FROM `users` ";

$run=$con->query($qry);

$num=$run->num\_rows;

while($rows = $run->fetch\_array()){

?>

<tr>

<td><?php echo $rows['UserName'];?></td>

<td><?php echo $rows['FirstName'];?></td>

<td><?php echo $rows['LastName'];?></td>

<td><?php echo $rows['Department'];?></td>

<td><?php echo $rows['age'];?></td>

<td><?php echo $rows['Password'];?></td>

<td><a class="btn btn-warning" href="edit.php?userName=<?php echo $rows['UserName'];?>">Edit</a></td>

<td><a class="btn btn-danger" href="delete.php?userName=<?php echo $rows['UserName'];?>">Delete</a></td>

</tr>

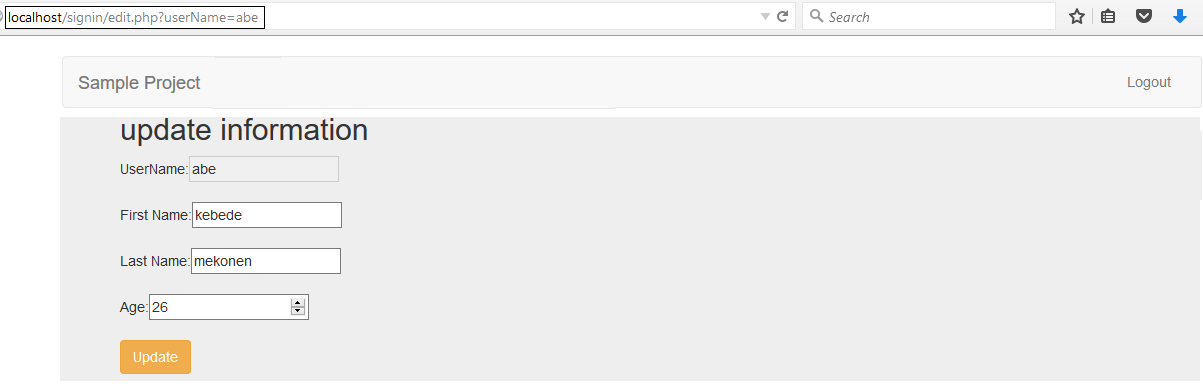
<?php } ?>

</tbody>

</table>

Toatal Users=<?php echo $num;?>

## **edit.php will look like this**



<?php

if(isset($\_GET["userName"])){

require 'db\_con.php';

$UN=$\_GET["userName"];

require 'db\_con.php';

$qry="SELECT \* FROM `users` WHERE `UserName`='$UN' ";

$run=$con->query($qry);

$rows=$run->fetch\_array();

$oldfname=$rows['FirstName'];

$oldlname=$rows['LastName'];

$oldage=$rows['age'];

?>

<h2>update information</h2>

<form name="frm1" method="post">

UserName:<input type="text" disabled="disabled" name="txtuname" value="<?php echo $UN;?>"/><br /><br />

First Name:<input type="text" name="txtfname" value="<?php echo $oldfname;?>"/><br /><br />

Last Name:<input type="text" name="txtlname" value="<?php echo $oldlname;?>"/><br /><br />

Age:<input type="number" name="txtage" value="<?php echo $oldage;?>"/><br /><br />

<input type="submit" name="btnupdate" class="btn btn-warning " value="Update"/>

</form>

<?php

if(isset($\_POST['btnupdate'])){

$newfname=$\_POST['txtfname'];

$newlname=$\_POST['txtlname'];

$newage=$\_POST['txtage'];

require 'db\_con.php';

$qry="UPDATE `users` SET `FirstName`='$newfname',`LastName`='$newlname',`age`='$newage' WHERE `UserName`='$UN' ";

$run=$con->query($qry);

if($run){

header('location:list.php');

}

}

}

?>

## **delete.php will look like this**

<?php

if(isset($\_GET["userName"])){

require 'db\_con.php';

$UN=$\_GET["userName"];

$qry="DELETE FROM `users` WHERE `UserName`='$UN' ";

$run=$con->query($qry);

if($run){

header('location:list.php');

}

}else{

header('location:list.php');

}

?>

## logout.php

We must have to unset our session and cookies before logout and redirect to login page.

<?php

session\_start();

if(!isset($\_SESSION["UserName"])){

header('location:index.php');

}else{

unset($\_SESSION["UserName"]);

session\_destroy();

$cookie\_name="UName";

setcookie($cookie\_name,"", time() - (86400 \* 30), "/");

header('location:index.php');

}

?>