PHP In 7 Days

Abhilash Sahoo

Contents

Chapter 1. Introduction

Chapter 2. PHP Variables and Strings

Chapter 3. PHP Operators

Chapter 4. PHP If Else and Switch Statements

Chapter 5. PHP Functions

Chapter 6. PHP Array

Chapter 7. PHP Looping

Chapter 8. PHP Forms and User Inputs

Chapter 9. PHP With MYSQL

Chapter 1. Introduction

PHP is a widely-used general-purpose scripting language that is especially suited for Web development and can be embedded into HTML. PHP is free and efficient alternative to competitors such as Microsoft's ASP

PHP is a powerful server-side scripting language for creating dynamic and interactive websites.

The PHP syntax is very similar to Perl and C. PHP is often used together with Apache (web server) on various operating systems. It also supports ISAPI and can be used with Microsoft's IIS on Windows.

1.1 What is PHP?

- PHP stands for **P**HP: **H**ypertext **P**reprocessor
- PHP is a server-side scripting language, like ASP
- PHP scripts are executed on the server
- PHP supports many databases (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, Generic ODBC, etc.)
- PHP is an open source software (OSS)
- PHP is free to download and use

1.2 What is a PHP File?

- PHP files may contain text, HTML tags and scripts
- PHP files are returned to the browser as plain HTML
- PHP files have a file extension of ".php", ".php3", or ".phtml"

1.3 What is MySQL?

- MySQL is a database server
- MySQL is ideal for both small and large applications
- MySQL supports standard SQL
- MySQL compiles on a number of platforms
- MySQL is free to download and use

1.4 PHP + MySQL

- PHP combined with MySQL are cross-platform (means that you can develop in Windows and serve on a Unix platform)

1.5 Why PHP?

- PHP runs on different platforms (Windows, Linux, Unix, etc.)
- PHP is compatible with almost all servers used today (Apache, IIS, etc.)
- PHP is FREE to download from the official PHP resource: www.php.net
- PHP is easy to learn and runs efficiently on the server side

1.6 Where to Start?

- Install an Apache server on a Windows or Linux machine
- Install PHP on a Windows or Linux machine
- Install MySQL on a Windows or Linux machine

1.7 What do You Need?

This book will not explain how to install PHP, MySQL, or Apache Server.

If your server supports PHP - you don't need to do anything! You do not need to compile anything or install any extra tools - just create some .php files in your web directory - and the server will parse them for you. Most web hosts offer PHP support.

However, if your server does not support PHP, you must install PHP. Below is a link to a good tutorial from PHP.net on how to install PHP5:

http://www.php.net/manual/en/install.php

1.8 Download PHP

Download PHP for free here: http://www.php.net/downloads.php

1.9 Download MySQL Database

Download MySQL for free here: http://www.mysql.com/downloads/index.html

1.10 Download Apache Server

Download Apache for free here: http://httpd.apache.org/download.cgi

1.11 Basic PHP Syntax

A PHP scripting block always starts with **<?php** and ends with **?>**. A PHP scripting block can be placed anywhere in the document.

On servers with shorthand support enabled you can start a scripting block with <? and end with ?>.

However, for maximum compatibility, we recommend that you use the standard form (<? php) rather than the shorthand form.

```
<?php
?>
```

A PHP file normally contains HTML tags, just like an HTML file, and some PHP scripting code.

Below, we have an example of a simple PHP script which sends the text "Hello World" to the browser:

```
<html>
<body>
<?php
echo "Hello World";
?>
</body>
</html>
```

Each code line in PHP must end with a semicolon. The semicolon is a separator and is used to distinguish one set of instructions from another.

There are two basic statements to output text with PHP: **echo** and **print**. In the example above we have used the echo statement to output the text "Hello World".

1.12 Comments in PHP

In PHP, we use // to make a single-line comment or /* and */ to make a large comment block.

```
<html>
<body>
<?php
//This is a comment
/*
This is
a comment
block
*/
?>
</body>
</html>
```

Chapter 2. PHP Variables and Strings

Variables are used for storing values, such as numbers, strings or function results, so that they can be used many times in a script.

2.1 Variables in PHP

Variables are used for storing values, like text strings, numbers or arrays.

When a variable is set it can be used over and over again in your script

All variables in PHP start with a $ sign symbol.

The correct way of setting a variable in PHP:

```
$var_name = value;
```

New PHP programmers often forget the $ sign at the beginning of the variable. In that case it will not work.

Let's try creating a variable with a string, and a variable with a number:

```
<?php
$txt = "Hello World!";
$number = 16;
?>
```

2.2 PHP is a Loosely Typed Language

In PHP a variable does not need to be declared before being set.

In the example above, you see that you do not have to tell PHP which data type the variable is.

PHP automatically converts the variable to the correct data type, depending on how they are set.

In a strongly typed programming language, you have to declare (define) the type and name of the variable before using it.

In PHP the variable is declared automatically when you use it.

2.3 Variable Naming Rules

- A variable name must start with a letter or an underscore "_"
- A variable name can only contain alpha-numeric characters and underscores (a-Z, 0-9, and _)
- A variable name should not contain spaces. If a variable name is more than one word, it should be separated with underscore ($my_string), or with capitalization ($myString)

PHP Strings

A string variable is used to store and manipulate a piece of text.

2.4 Strings in PHP

String variables are used for values that contain character strings.

In this book we are going to look at some of the most common functions and operators used to manipulate strings in PHP.

After we create a string we can manipulate it. A string can be used directly in a function or it can be stored in a variable.

Below, the PHP script assigns the string "Hello World" to a string variable called $txt:

```
<?php
$txt="Hello World";
echo $txt;
?>
```

The output of the code above will be:

```
Hello World
```

Now, lets try to use some different functions and operators to manipulate our string.

2.5 The Concatenation Operator

There is only one string operator in PHP.

The concatenation operator (.) is used to put two string values together.

To concatenate two variables together, use the dot (.) operator:

```
<?php
$txt1="Hello World";
$txt2="1234";
echo $txt1 . " " . $txt2;
?>
```

The output of the code above will be:

```
Hello World 1234
```

If we look at the code above you see that we used the concatenation operator two times. This is because we had to insert a third string.

Between the two string variables we added a string with a single character, an empty space, to separate the two variables.

2.6 Using the strlen() function

The strlen() function is used to find the length of a string.

Let's find the length of our string "Hello world!":

```
<?php
echo strlen("Hello world!");
?>
```

The output of the code above will be:

```
12
```

The length of a string is often used in loops or other functions, when it is important to know when the string ends. (i.e. in a loop, we would want to stop the loop after the last character in the string)

2.7 Using the strpos() function

The strpos() function is used to search for a string or character within a string.

If a match is found in the string, this function will return the position of the first match. If no match is found, it will return FALSE.

Let's see if we can find the string "world" in our string:

```
<?php
echo strpos("Hello world!","world");
?>
```

The output of the code above will be:

```
6
```

As you see the position of the string "world" in our string is position 6. The reason that it is 6, and not 7, is that the first position in the string is 0, and not 1.

Chapter 3. PHP Operators

Operators are used to operate on values.

3.1 PHP Operators

This section lists the different operators used in PHP.

Arithmetic Operators

Operator	Description	Example	Result
+	Addition	x=2 x+2	4
-	Subtraction	x=2 5-x	3
*	Multiplication	x=4 x*5	20
/	Division	15/5 5/2	3 2.5
%	Modulus (division remainder)	5%2 10%8 10%2	1 2 0
++	Increment	x=5 x++	x=6
--	Decrement	x=5 x--	x=4

3.2 Assignment Operators

Operator	Example	Is The Same As
=	x=y	x=y
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
.=	x.=y	x=x.y
%=	x%=y	x=x%y

3.3 Comparison Operators

Operator	Description	Example
==	is equal to	5==8 returns false
!=	is not equal	5!=8 returns true
>	is greater than	5>8 returns false
<	is less than	5<8 returns true
>=	is greater than or equal to	5>=8 returns false
<=	is less than or equal to	5<=8 returns true

3.4 Logical Operators

Operator	Description	Example
&&	and	x=6 y=3 (x < 10 && y > 1) returns true
||	or	x=6 y=3 (x==5 || y==5) returns false
!	not	x=6 y=3 !(x==y) returns true

Chapter 4. PHP If Else and Switch Statements

The if, elseif and else statements in PHP are used to perform different actions based on different conditions.

4.1 Conditional Statements

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

- **if...else statement** - use this statement if you want to execute a set of code when a condition is true and another if the condition is not true
- **elseif statement** - is used with the if...else statement to execute a set of code if **one** of several condition are true

4.2 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>
<body>
<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
else
  echo "Have a nice day!";
?>
</body>
</html>
```

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

```
<html>
<body>
```

```
<?php
$d=date("D");
if ($d=="Fri")
  {
  echo "Hello!<br />";
  echo "Have a nice weekend!";
  echo "See you on Monday!";
  }
?>
</body>
</html>
```

4.3 The ElseIf 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!":

```
<html>
<body>
<?php
$d=date("D");
if ($d=="Fri")
  echo "Have a nice weekend!";
elseif ($d=="Sun")
  echo "Have a nice Sunday!";
else
  echo "Have a nice day!";
?>
</body>
</html>
```

Switch Statements

The Switch statement in PHP is used to perform one of several different actions based on one of several different conditions.

4.4 The Switch Statement

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..elseif..else code.

Syntax

```
switch (expression)
{
case label1:
  code to be executed if expression = label1;
  break;
case label2:
  code to be executed if expression = label2;
  break;
default:
  code to be executed
  if expression is different
  from both label1 and label2;
}
```

Example

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

```
<html>
<body>
<?php
switch ($x)
{
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";
}
?>
</body>
</html>
```

Chapter 5. PHP Functions

The real power of PHP comes from its functions.

In PHP - there are more than 700 built-in functions available.

5.1 PHP Functions

In this book we will show you how to create your own functions.

5.2 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:

```
<html>
<body>
<?php
function writeMyName()
  {
  echo "Kai Jim Refsnes";
  }
writeMyName();
?>
</body>
</html>
```

5.3 Use a PHP Function

Now we will use the function in a PHP script:

```
<html>
<body>
<?php
function writeMyName()
  {
  echo "Kai Jim Refsnes";
  }
echo "Hello world!<br />";
echo "My name is ";
writeMyName();
echo ".<br />That's right, ";
```

```
writeMyName();
echo " is my name.";
?>
</body>
</html>
```

The output of the code above will be:

```
Hello world!
My name is Kai Jim Refsnes.
That's right, Kai Jim Refsnes is my name.
```

5.4 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:

```
<html>
<body>
<?php
function writeMyName($fname)
  {
  echo $fname . " Refsnes.<br />";
  }
echo "My name is ";
writeMyName("Kai Jim");
echo "My name is ";
writeMyName("Hege");
echo "My name is ";
writeMyName("Stale");
?>
</body>
</html>
```

The output of the code above will be:

```
My name is Kai Jim Refsnes.
My name is Hege Refsnes.
My name is Stale Refsnes.
```

Example 2

The following function has two parameters:

```
<html>
<body>
<?php
function writeMyName($fname,$punctuation)
  {
```

```
  echo $fname . " Refsnes" . $punctuation . "<br />";
  }
echo "My name is ";
writeMyName("Kai Jim",".");
echo "My name is ";
writeMyName("Hege","!");
echo "My name is ";
writeMyName("Ståle","...");
?>
</body>
</html>
```

The output of the code above will be:

```
My name is Kai Jim Refsnes.
My name is Hege Refsnes!
My name is Ståle Refsnes...
```

5.5 PHP Functions - Return values

Functions can also be used to return values.

Example

```
<html>
<body>
<?php
function add($x,$y)
  {
  $total = $x + $y;
  return $total;
  }
echo "1 + 16 = " . add(1,16);
?>
</body>
</html>
```

The output of the code above will be:

```
1 + 16 = 17
```

Chapter 6. PHP Arrays

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

6.1 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

6.2 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("Peter","Quagmire","Joe");
```

Example 2

In this example we assign the ID key manually:

```
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
```

The ID keys can be used in a script:

```
<?php
$names[0] = "Peter";
$names[1] = "Quagmire";
$names[2] = "Joe";
echo $names[1] . " and " . $names[2] .
" are ". $names[0] . "'s neighbors";
?>
```

The code above will output:

```
Quagmire and Joe are Peter's neighbors
```

6.3 Associative Arrays

An associative array, each ID key is associated with a value.

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";
$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.";
?>
```

The code above will output:

```
Peter is 32 years old.
```

6.4 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
  )
)
```

Example 2

Lets try displaying a single value from the array above:

```
echo "Is " . $families['Griffin'][2] .
" a part of the Griffin family?";
```

The code above will output:

```
Is Megan a part of the Griffin family?
```

Chapter 7. PHP Looping

Looping statements in PHP are used to execute the same block of code a specified number of times.

7.1 Looping

Very often when you write code, you want the same block of code to run a number of times. You can use looping statements in your code to perform this.

In PHP we have the following looping statements:

- **while** - loops through a block of code if and as long as a specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as a special condition is true
- **for** - loops through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

7.2 The while Statement

The while statement will execute a block of code **if and as long as** a condition is true.

Syntax

```
while (condition)
code to be executed;
```

Example

The following example demonstrates a loop that will continue to run as long as the variable i is less than, or equal to 5. i will increase by 1 each time the loop runs:

```
<html>
<body>
<?php
$i=1;
while($i<=5)
  {
  echo "The number is " . $i . "<br />";
  $i++;
  }
?>
</body>
</html>
```

7.3 The do...while Statement

The do...while statement will execute a block of code **at least once** - it then will repeat the loop **as long as** a condition is true.

Syntax

```
do
```

```
{
code to be executed;
}
while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 5:

```
<html>
<body>
<?php
$i=0;
do
  {
  $i++;
  echo "The number is " . $i . "<br />";
  }
while ($i<5);
?>
</body>
</html>
```

7.4 The for Statement

The for statement is used when you know how many times you want to execute a statement or a list of statements.

Syntax

```
for (initialization; condition; increment)
{
  code to be executed;
}
```

Note: The for statement has three parameters. The first parameter initializes variables, the second parameter holds the condition, and the third parameter contains the increments required to implement the loop. If more than one variable is included in the initialization or the increment parameter, they should be separated by commas. The condition must evaluate to true or false.

Example

The following example prints the text "Hello World!" five times:

```
<html>
<body>
<?php
for ($i=1; $i<=5; $i++)
{
  echo "Hello World!<br />";
}
?>
</body>
</html>
```

7.5 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 as value)
{
    code to be executed;
}
```

Example

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

```
<html>
<body>
<?php
$arr=array("one", "two", "three");
foreach ($arr as $value)
{
  echo "Value: " . $value . "<br />";
}
?>
</body>
</html>
```

Chapter 8. PHP Forms and User Input

The PHP $_GET and $_POST variables are used to retrieve information from forms, like user input.

8.1 PHP Form Handling

The most important thing to notice when dealing with HTML forms and PHP is that any form element in an HTML page will **automatically** be available to your PHP scripts.

Form example:

```
<html>
<body>
<form action="welcome.php" method="post">
Name: <input type="text" name="name" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
</body>
</html>
```

The example HTML page above contains two input fields and a submit button. When the user fills in this form and click on the submit button, the form data is sent to the "welcome.php" file.

The "welcome.php" file looks like this:

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old.
</body>
</html>
```

A sample output of the above script may be:

```
Welcome John.
You are 28 years old.
```

The PHP $_GET and $_POST variables will be explained in the next chapters.

8.2 Form Validation

User input should be validated whenever possible. Client side validation is faster, and will reduce server load.

However, any site that gets enough traffic to worry about server resources, may also need to worry about site security. You should always use server side validation if the form accesses a database.

A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

PHP $_GET

The $_GET variable is used to collect values from a form with method="get".

8.3 The $_GET Variable

The $_GET variable is an array of variable names and values sent by the HTTP GET method.

The $_GET variable is used to collect values from a form with method="get". Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and it has limits on the amount of information to send (max. 100 characters).

Example

```
<form action="welcome.php" method="get">
Name: <input type="text" name="name" />
Age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL sent could look something like this:

```
http://www.w3schools.com/welcome.php?name=Peter&age=37
```

The "welcome.php" file can now use the $_GET variable to catch the form data (notice that the names of the form fields will automatically be the ID keys in the $_GET array):

```
Welcome <?php echo $_GET["name"]; ?>.<br />
You are <?php echo $_GET["age"]; ?> years old!
```

8.4 Why use $_GET?

Note: When using the $_GET variable all variable names and values are displayed in the URL. So this method should not be used when sending passwords or other sensitive information! However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

Note: The HTTP GET method is not suitable on large variable values; the value cannot exceed 100 characters.

8.5 The $_REQUEST Variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Example

```
Welcome <?php echo $_REQUEST["name"]; ?>.<br />
You are <?php echo $_REQUEST["age"]; ?> years old!
```

PHP $_POST

The $_POST variable is used to collect values from a form with method="post".

8.6 The $_POST Variable

The $_POST variable is an array of variable names and values sent by the HTTP POST method.

The $_POST variable is used to collect values from a form with method="post". Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

Example

```
<form action="welcome.php" method="post">
Enter your name: <input type="text" name="name" />
Enter your age: <input type="text" name="age" />
<input type="submit" />
</form>
```

When the user clicks the "Submit" button, the URL will not contain any form data, and will look something like this:

```
http://www.w3schools.com/welcome.php
```

The "welcome.php" file can now use the $_POST variable to catch the form data (notice that the names of the form fields will automatically be the ID keys in the $_POST array):

```
Welcome <?php echo $_POST["name"]; ?>.<br />
You are <?php echo $_POST["age"]; ?> years old!
```

8.7 Why use $_POST?

- Variables sent with HTTP POST are not shown in the URL
- Variables have no length limit

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

8.8 The $_REQUEST Variable

The PHP $_REQUEST variable contains the contents of both $_GET, $_POST, and $_COOKIE.

The PHP $_REQUEST variable can be used to get the result from form data sent with both the GET and POST methods.

Example

```
Welcome <?php echo $_REQUEST["name"]; ?>.<br />
You are <?php echo $_REQUEST["age"]; ?> years old!
```

Chapter 9. PHP With MYSQL

MySQL is the most popular open source database server.

What is MySQL?

MySQL is a database. A database defines a structure for storing information.

In a database, there are tables. Just like HTML tables, database tables contain rows, columns, and cells.

Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".

Database Tables

A database most often contains one or more tables. Each table has a name (e.g. "Customers" or "Orders"). Each table contains records (rows) with data.

Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The table above contains three records (one for each person) and four columns (LastName, FirstName, Address, and City).

Queries

A query is a question or a request.

With MySQL, we can query a database for specific information and have a recordset returned.

Look at the following query:

```
SELECT LastName FROM Persons
```

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

LastName
Hansen
Svendson
Pettersen

Download MySQL Database

If you don't have a PHP server with a MySQL Database, you can download MySQL for free here: http://www.mysql.com/downloads/index.html

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, Google). Look at http://www.mysql.com/customers/ for an overview of companies that use MySQL.

PHP MySQL Connect to a Database

The free MySQL Database is very often used with PHP.

Connecting to a MySQL Database

Before you can access and work with 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);
```

Parameter	Description
servername	Optional. Specifies the server to connect to. Default value is "localhost:3306"
username	Optional. Specifies the username to log in with. Default value is the name of the user that owns the server process
password	Optional. Specifies the password to log in with. Default is ""

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
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// some code
?>
```

Closing a Connection

The connection will be closed as soon as the script ends. To close the connection before, use the mysql_close() function.

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// some code
mysql_close($con);
?>
```

PHP MySQL Create Database and Tables

A database holds one or multiple tables.

Create a Database

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

Syntax

```
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

In the following example we create a database called "my_db":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$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 database 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.

Example

The following example shows how you can create a table named "person", with three columns. The column names will be "FirstName", "LastName" and "Age":

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
// Create database
if (mysql_query("CREATE DATABASE my_db",$con))
  {
  echo "Database created";
  }
else
  {
```

```
  echo "Error creating database: " . mysql_error();
  }
// Create table in my_db database
mysql_select_db("my_db", $con);
$sql = "CREATE TABLE person
(
FirstName varchar(15),
LastName varchar(15),
Age int
)";
mysql_query($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).

MySQL Data Types

Below are the different MySQL data types that can be used:

Numeric Data Types	Description
int(size) smallint(size) tinyint(size) mediumint(size) bigint(size)	Hold integers only. The maximum number of digits can be specified in the size parameter
decimal(size,d) double(size,d) float(size,d)	Hold numbers with fractions. The maximum number of digits can be specified in the size parameter. The maximum number of digits to the right of the decimal is specified in the d parameter

Textual Data Types	Description
char(size)	Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis
varchar(size)	Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis
tinytext	Holds a variable string with a maximum length of 255 characters
text	Holds a variable string with a maximum length of

blob	65535 characters
mediumtext mediumblob	Holds a variable string with a maximum length of 16777215 characters
longtext longblob	Holds a variable string with a maximum length of 4294967295 characters

Date Data Types	**Description**
date(yyyy-mm-dd) datetime(yyyy-mm-dd hh:mm:ss) timestamp(yyyymmddhhmmss) time(hh:mm:ss)	Holds date and/or time

Misc. Data Types	**Description**
enum(value1,value2,ect)	ENUM is short for ENUMERATED list. Can store one of up to 65535 values listed within the ( ) brackets. If a value is inserted that is not in the list, a blank value will be inserted
set	SET is similar to ENUM. However, SET can have up to 64 list items and can store more than one choice

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 primary key field is always indexed. There is no exception to this rule! You must index the primary key field so the database engine can quickly locate rows based on the key's value.

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 person
(
```

```
personID int NOT NULL AUTO_INCREMENT,
PRIMARY KEY(personID),
FirstName varchar(15),
LastName varchar(15),
Age int
)";
mysql_query($sql,$con);
```

PHP MySQL Insert Into

The INSERT INTO statement is used to insert new records into a database table.

Insert Data Into a Database Table

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

Syntax

```
INSERT INTO table_name
VALUES (value1, value2,....)
```

You can also specify the columns where you want to insert the data:

```
INSERT INTO table_name (column1, column2,...)
VALUES (value1, value2,....)
```

Note: SQL statements are not case sensitive. INSERT INTO is the same as insert into.

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 "Person", 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 "Person" table:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
mysql_query("INSERT INTO person (FirstName, LastName, Age)
VALUES ('Peter', 'Griffin', '35')");
mysql_query("INSERT INTO person (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 "Person" table.

Here is the HTML form:

```
<html>
<body>
<form action="insert.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 database table.

Below is the code in the "insert.php" page:

```
<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
  {
  die('Could not connect: ' . mysql_error());
  }
mysql_select_db("my_db", $con);
$sql="INSERT INTO person (FirstName, LastName, Age)
VALUES
('$_POST[firstname]','$_POST[lastname]','$_POST[age]')";
if (!mysql_query($sql,$con))
  {
  die('Error: ' . mysql_error());
  }
echo "1 record added";
mysql_close($con)
?>
```