The Ultimate Guide to PHP Programming!

LEARN PHP

Learn PHP Programming in 4 hours!
PHP for Beginners - Smart and Easy ways to learn PHP & MYSQL

Barry Page

PHP Programming

PHP Crush Course! Learn PHP Programming in 4 hours! PHP for Beginners - Smart and Easy ways to learn PHP & MySQL

Book Description

“Learn PHP in 4 Hours!” promises to give you complete introductory knowledge to PHP and MySQL. You will be able to get started on your journey to building awesome dynamic websites that help you achieve your dreams in no time.

This book contains definitions that are straightforward, examples that are short and sweet and explanations that ensure mastery of the basics very quickly.

You will be able to blast through this book while gaining deep knowledge that prepares you to tackle the advanced features of both PHP and MySQL while simultaneously being able to do something very useful with the best practices. This is an amazing educational book!

What is in the book?

Introduction to PHP Programming

Chapter 1 – Hour 1: Installation and PHP

We discuss installation of a web server package and show how to use it to run PHP script. We also cover some of the basics of PHP

Chapter 2 – Hour 2: More PHP Basics

We dive deeper into the basics of PHP

Chapter 3 – Hour 3: MySQL Basics

We discuss how to access the command line to enter MySQL commands and use them to work with data in the database

Chapter 4 – Hour 4: PHP and MySQL

We consolidate the knowledge and use both PHP and MySQL to create a very simple website with dynamic content

Let’s get started!

Table of Content

Introduction

“You cannot open a book without learning something.” - Confucius

People learn PHP for many reasons. Every language has a bit of a learning curve and one can get bogged down in two much theory that it can make it unappealing because the task ahead seems odious. Maybe you want to build a simple blog or something as advanced as a massive online e-commerce site with thousands of products up for sale. It is hard to know where to start getting the needed knowledge.

If you are looking for a way to get going fast in order to make you visions a reality then you have come to the right place! This books aims at getting you started on your journey to PHP and MySQL mastery in 4 hours! It does this by telling you the most important stuff you need to know to get started quickly for the absolute beginner, in either PHP or MySQL, so you get to doing useful stuff as quickly as possible.

This is not a comprehensive guide to PHP and MySQL that covers all the aspects but a crash course that gives you a practical guide on how to do really cool basic things with PHP and MySQL on their own and together. The two can get pretty advanced but with the knowledge in this book, you should be able to get that prototype up and running as you tackle the more advanced topics.

A little knowledge of HTML and CSS will be needed before because, really, PHP is just a language that is used to produce web pages and its final output is HTML. We will go over all the basics of PHP and MySQL and finally put it all together and build simple web site where you will be able to store a records of your favourite artists using the dynamic duo.

The basics are all you need to get going in 4 hours (and this book, of course) is all you need.

But what is PHP or MySQL and why do you need it to accomplish this previously impossible task?

A Short Introduction to PHP

PHP stands for Hypertext Processor. It is a server side scripting language that is used to produce dynamic web content among other things. This means that everything PHP does is not done on the computer that is used to view the web pages, called the client, but a specialised computer called a server that houses all the webpages that you are viewing on the client machine.

Being a scripting language means that PHP can’t really be used to make standalone applications that run on your computer as it is designed to only do things when an event happens within web pages like; clicking on a link or getting form data when a web form is sent.

You can embed PHP into your HTML web pages and produce pages that are customised according to the information you get from your users. This ends up making webpages more exciting rather than if they were just static.PHP can be used to produce web pages because its final output is HTML.

It is very efficient in the server side, widely used, cross-platform (runs on Windows, Linux, UNIX, Mac OS X, etc.) and easier to learn than any other fully featured language like C, Java or Perl.

PHP is also very powerful. If you ever doubt its power, just look carefully at the link in your browser next time you decide to log on to Facebook. You will notice that the largest social networking site on the planet is powered by PHP! Even Wikipedia (That site which tells you everything you need to know) is powered by PHP. That is the power that is behind PHP, and it makes it a good alternative to its competitors such as Microsoft's Active Server Pages (ASP).

It also has an active community that can really help you in a bind and always active.

According to the http://www.w3schools.com PHP page; this is what PHP will allow you to do:

- **PHP can generate dynamic page content**
- **PHP can create, open, read, write, delete, 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 (that is why it is a perfect match for MySQL)**
- **PHP can be used to control user-access**
- **PHP can encrypt data**

A Short Introduction to MySQL

MySQL is a popular open source relational database that uses SQL as its base. It enables you to store, retrieve and manipulate data. It is a highly efficient and scalable solution that is also cost effective.

Many web based programs (including computer programs) use databases to store their information. Things like social applications need a place to store user information, pictures, posts, comments etc. that is where a database like MySQL comes in handy and makes storing and retrieving large amounts of information a breeze.

SQL, often pronounced as sequel, stands for Structured Query Language. It is simply a language designed for management of data by databases like MySQL. They use SQL to send and retrieve data using specialised commands that really aren’t all that hard to learn. And most importantly, for this book, you can embed SQL into programming languages like PHP.

Also, MySQL is open source and is available for free and can be changed by modifying the source code, if particular features don’t suit your fantacy.

What We Will Cover in This Book

This book is separated into four main chapters or hours that help you get to the heart of PHP and SQL and send you on your way to being a master in no time.

In the first hour we will spend time installing the much needed web server package called **XAMPP**. I recommend that you download it here:

https://www.apachefriends.org/index.html before we start that chapter. This will enable us to run examples throughout this book. Then we shall pick a text editor for our PHP code. Then we will look at how to run PHP scripts and, finally, we shall cover some of the basics of PHP, such as comments, variables, constants and operators.

The examples in this book can be typed out or, to save time, copied and then pasted into your code editor or command line interface.

In the second hour, we shall continue with PHP and cover conditional statements, Loops, Arrays and functions. After this chapter, you should consider yourself well acquainted with PHP to tackle further advanced topics.

You can take a break after this part and let it all sink in and make another cup of coffee.

In the third hour we tackle MySQL and its commands, but first we learn how to access the command line. Then we move on to the commands that let us create and delete databases; create, alter and delete tables; insert, select, update and delete data in the tables.

Finally, in the final hour, we put PHP and MySQL together and create a simple website to show you how these two work together hand in hand.

Let us get to it. Grab your first cup of coffee!

Chapter 1 - Hour 1: Installing and PHP Basics

“The way to get started is to take the first step with dream-powered optimism.” - Debasish Mridha

This chapter we will cover:

- Installing a web server
- Installing a text editor
- Starting Apaches to run PHP scripts
- Creating and running a PHP script
- PHP basics (comment, variables, operators, constants, strings)

Bear with me a little. I know that the first chapter of almost every programming book piles on the theory to make sure you know what you are getting yourself into. I won’t lie; even in this short book, we still have to go through it a little. Just a little bit of it to cover some basics of PHP. But that is for later on during the hour. For now you need to install the server and choose an appropriate text editor to use when writing PHP code!

Installing a Web Server

Your first real step to PHP mastery has led you here. You can't just run PHP in a browser like you do with HTML. To run PHP, you are going to need a server installed on your local machine in order for the browser to parse PHP scripts and run MySQL queries to the database to return results. Alternatively, you can use a host that supports PHP and MySQL so that they can upload it to their server and you can test and deploy your dream site right there. But I'm guessing you really want to test it locally and quickly. So, we will install a local server that makes your very own computer become a test server.

There are many servers you can install on your local machine such as XAMP, WAMP (for windows), and MAMP for Mac OS X, etc., but this book covers XAMP. It is a free open source cross platform web server package available for Windows, Mac and Linux.

Go to this website

https://www.apachefriends.org/index.html

and download the latest version of XAMP for windows (the same steps can be easily translated to Mac OSX or Linux). Once the download is finished, locate the file on your local machine and double-click it to install it.

Simple! Now PHP and MySQL are ready for use.

NOTE: If you have an antivirus running at this time, it may interfere with the process. You might want to switch it off until the server is done installing to avoid any potential problems.

Installing a Text Editor

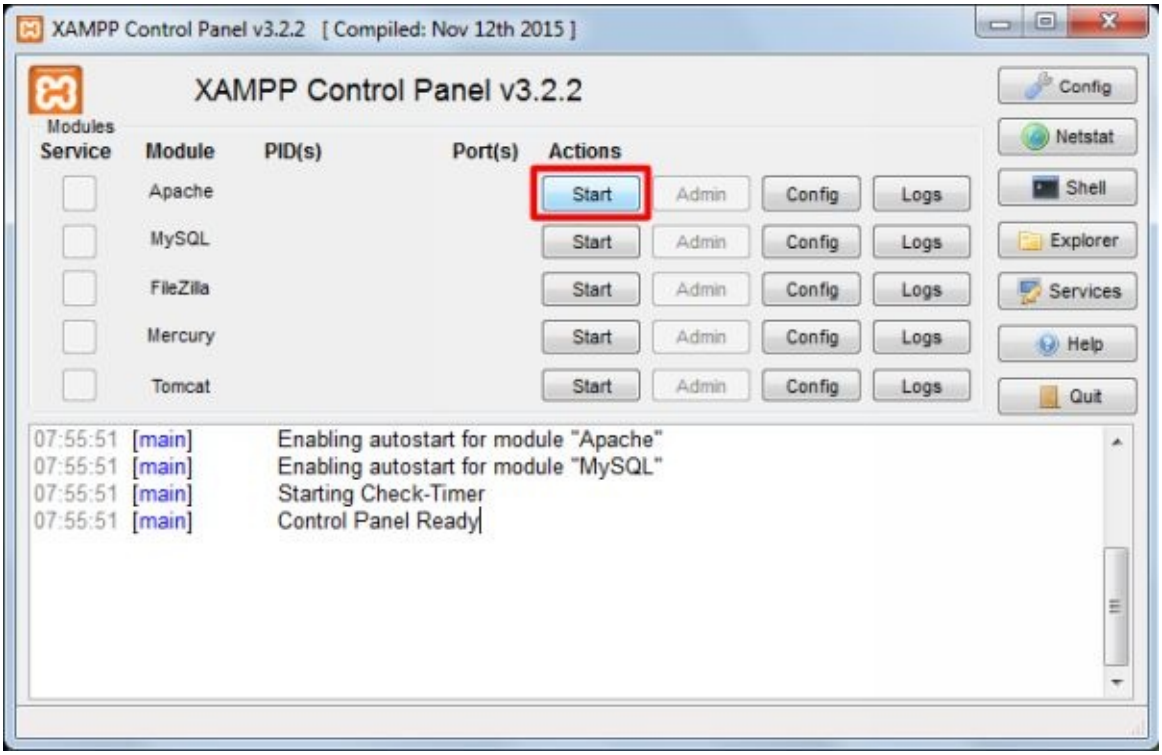
Now, you will need a text editor to write some beautiful PHP. Ideally, any text editor would do, even the default notepad on windows or terminal for Mac OS X can be used when writing PHP. We will use notepad++ because it has some nifty features which can contribute towards writing clean and efficient code in a fast way, such as syntax highlighting and auto complete.

Plus it’s free!

This text editors will save you time and a world of pain by just downloading it here Installing a Text Editor and installing it. Just remember that any text will do. So even if you don’t have notepad++, you can use whatever you like.

Starting Apache to run PHP scripts

Now that you have installed XAMPP, you need to start the apache web service in order for the browser to be able to run PHP scripts. To do this, open up the Control Panel of XAMPP by double clicking on its icon. By default, it should start Apache automatically, but you can start it manually by clicking “Start” as in the picture below:



NOTE: Some other applications installed on your computer can cause Apache not to start because they are using the same port Apache is configured to e.g. Skype. If that is the case, just stop those services and start Apache again.

Creating Your First PHP Script

This part is fairly easy to do. All you have to do is follow these steps:

Step 1 - Create the PHP file

Open up your text editor and create a regular text file and place the following code into it:

```
<!DOCTYPE html>
<html>
        <body>
           <?php
                        echo “My first PHP script!”;
                ?>
   </body>
</html>
```

The file still looks like your basic HTML file except that you embed PHP code between tags **<?php** and **?>**. The **echo** statement just outputs everything that comes after it to the screen. In our case, this will be “My first PHP script!”. We shall see this in action in a moment.

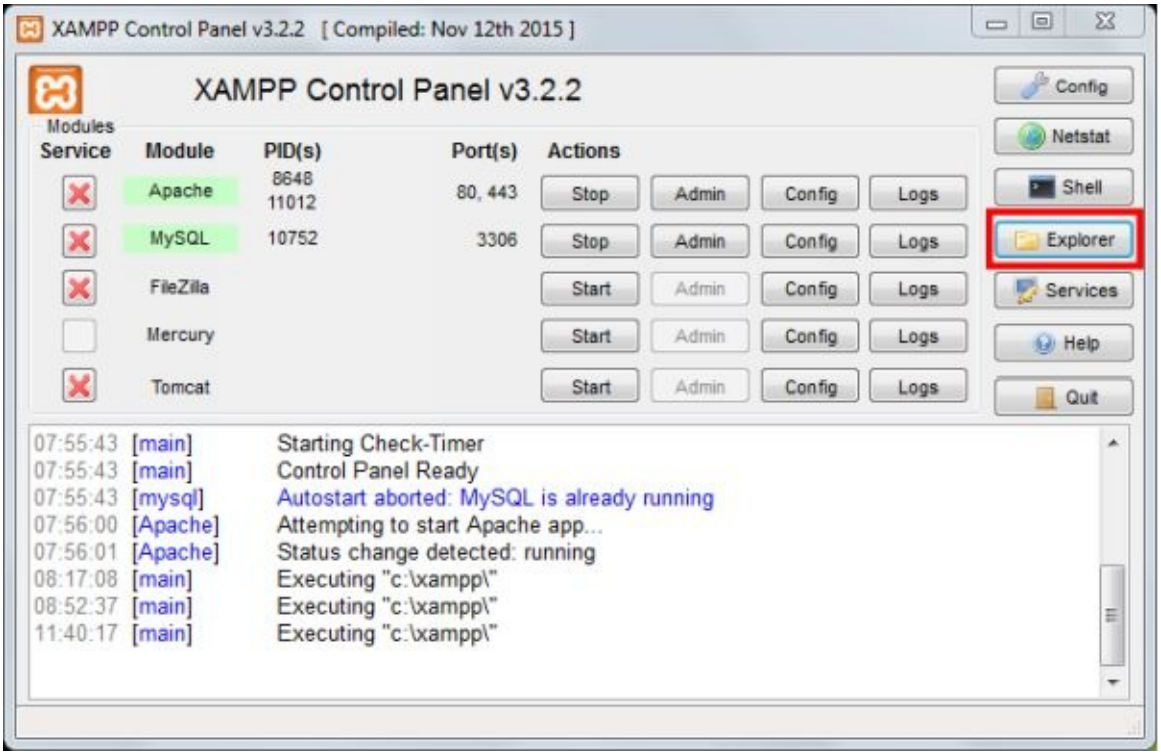
And all you have to remember is that every PHP statement ends with a semicolon (Except for comments. We will look at those later in the hour).

Save the file as “example.php”.

Step 2 - Place it in the Root Directory of XAMPP

Grab the file “example.php” that you just created and place it in the root directory of your server in the folder named “htdocs”, and Whalla! You are ready to run it in your browser.

TIP: To find the root directory, you can look for it on your local machine or you could just open up the XAMPP Control Panel and click the “Explorer” button, as shown in the picture below, to bring up the root folder and then locate the htdocs folder and place the file in there:



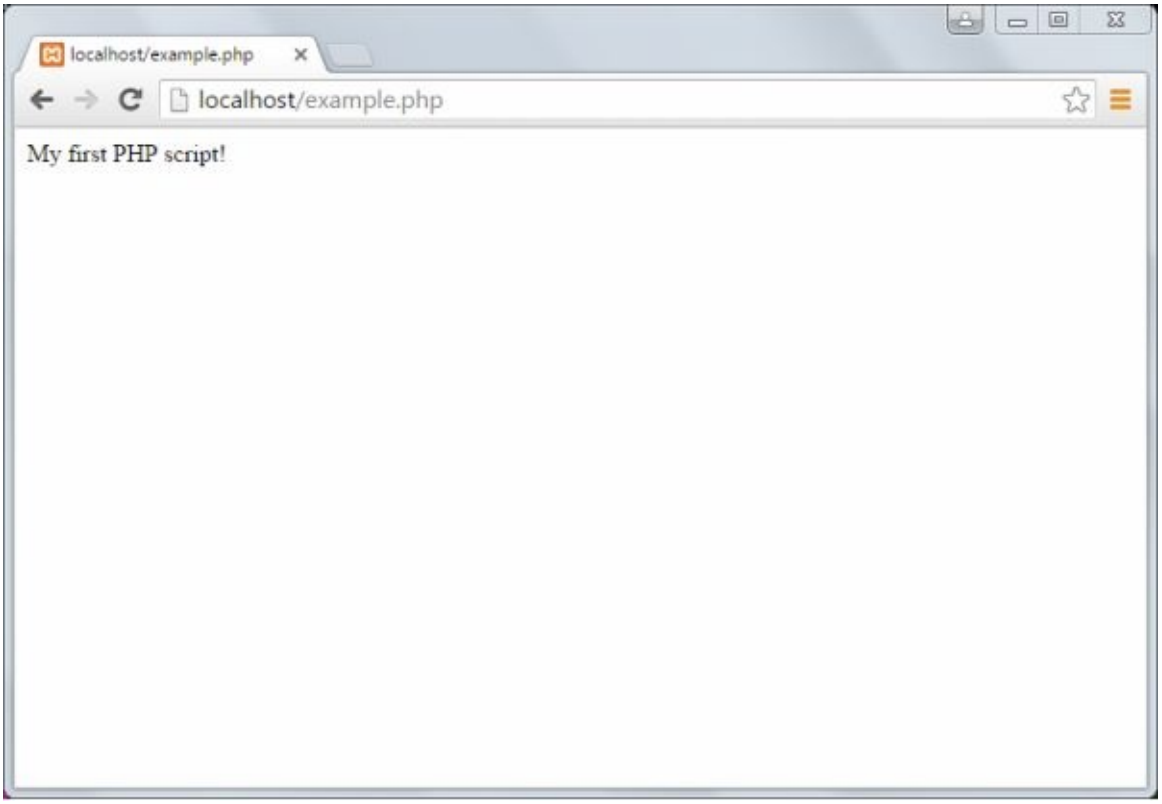
NOTE: The folder to place your PHP files will be different with each server. For example; you place them in the folder named www if you are using WAMP.

Step 3 - Run It in the Browser

Open up your browser (Google Chrome or Firefox are recommended) and type the following:

http://localhost/example.php

Press Enter and you should see the following window below:



There you go! If you follow those steps, you should be able to run all other scripts created in this chapter.

PHP Basics

Comments

Want to leave messages for your future self or other members of your team? Then comments save you the trouble of having to remember your train of thought and make for an excellent way to document what your thoughts were at the time. They can also be used to exclude certain pieces of code all together.

// This is a single line comment

 # This is another single line comment

**/*
This is a comment block. Everything in here will be ignored and can span multiple lines.
*/**

Variables

To build your dream website, you will need data and a way to store this data. It can be a name, a number, a date or a picture. PHP offers a way for you to store them in memory by using a variable.

Each variable in PHP is preceded by the $ sign followed by the variable name without any spaces in between the sign and variable name:

$myInt = 10;

Let us see some of this in action with the following code:

```
<!DOCTYPE html>
<html>
        <body>
                <?php
                        $txt = “My second PHP script!”;
                        echo $txt;
                ?>
        </body>
</html>
```

If you run this script in the browser, it should out put the following:

“My second PHP script!”;

Few more things to remember about variables:

- They are case sensitive. This means $name and $naMe are two different variables
- They must always start with a letter or underscore
- They cannot start with numbers
- They can only contain alpha numeric values

PHP is a loosely type language and therefore you don’t need to define the data types for variables. PHP will automatically convert the variable to the correct data type depending on the value assigned to it.

Here is a list of the common data types you will work with:

- Integer – Whole numbers
- Float – Decimal numbers
- String – String or characters
- Boolean – true or false
- Array – Multiple items
- Object – An Object defined by a class

You can achieve the same result as the example above by using variables:

```
<!DOCTYPE html>
<html>
	<body>
		<?php
			$txt = “third”;
			echo “My ” .  $txt .” PHP script!”;
		?>
	</body>
</html>
```

In this example we are joining two strings together using the period (.) as it is used to concatenate two strings together and outputting the concatenated string to the screen.

Constants

Constants are like variables except that their values cannot be changed or undefined.

When define a constant we use define() method followed by parameters in the parenthesis that give it a name and a value. The following example shows how to define a constant:

```
<?php
        define(“MESSAGE”, “Happy Learning!”);  //function that defines a constant
        echo MESSAGE;
?>
```

Output:

Happy Learning!

Unlike variables, you can actually specify whether you want the constant to be case-insensitive or not by adding one more parameter, that takes a Boolean, after the value is defined (default is **false**):

```
<?php
        define(“MESSAGE”, “Happy Learning!”,  true,);
        echo message;
?>
```

Some rules to remember about constants:

- **They must start with a letter and an underscore (they don’t need to start with the $ sign)**
- **They global across the entire script automatically**

Operators

PHP uses operators to perform various operations of variables and values. We shall now look at the operators used by PHP in groups.

Arithmetic Operators

These are operators that perform mathematical operations on values and return calculated results:

EXAMPLE	OPERATOR	RESULT
$x + $y	**Addition**	**Sum of $x and $y**
$x - $y	**Subtraction**	**The difference between $x and $y**
$x * $y	**Multiplication**	**The Product of $x and $y**
$x / $y	**Division**	**The dividend of $x and $y**
$x / $y	**Modulus**	**The remainder of $x divided by $y**
$x ** $y	**Exponent**	**Result of $x to $y'th power (New to PHP 5.6)**

Assignment Operators

These operators are used to set values to variables:

ASSIGNMENT	SAME AS:	DESCRIPTION
$x = $y	**$x = $y**	**Gets the value of $y and sets it to $x**
$x += $y	**$x = $x + $y**	**Gets the sum of $x and $y and sets the result into $x**
$x -= $y	**$x = $x - $y**	**Gets the difference of $x and $y and sets the result into $x**
$x *= $y	**$x = $x * $y**	**Gets the product of $x and $y and sets the result into $x**
$x /= $y	**$x = $x / $y**	**Gets the dividend of $x and $y and sets the result into $x**
$x %= $y	**$x = $x % $y**	**Gets the remainder of $x divided by $y and sets the result into $x**

Comparison Operators

Comparison operators in PHP are used to compare two values:

EXAMPLE	NAME	RESULT
$x == $y	**Equal**	**Returns true if both $x and $y are equal**
$x === $y	**Identical**	**Returns true if both $x and $y are equal, and have the same data type**
$x != $y	**Not equal**	**Returns true if both $x and $y are not equal**
$x <> $y	**Not equal**	**Returns true if both $x and $y are not equal**
$x !== $y	**Not identical**	**Returns true if both $x and $y are not equal, or if they don't have the same data type**
$x > $y	**Greater than**	**Returns true if $x is greater than $y**
$x < $y	**Less Than**	**Returns true if $x is less than $y**
$x >= $y	**Greater than or equal to**	**Returns true if $x is greater than or equal to $y**
$x <= $y	**Less Than or equal to**	**Returns true if $x is less than or equal $y**

Logical Operators

Logical operators are what we use to combine conditional statements (which we will cover in the next hour) in order to get a true or false result:

EXAMPLE	NAME	RESULT
$x && $y	**And**	**Returns true if both $x and $y are true**
$x || $y	**Or**	**Returns true either $x or $y are true**
!$x	**Not**	**Returns true if $x is not true**
$x and $y	**And**	**Returns true if both $x and $y are true**
$x or $y	**Or**	**Returns true either $x or $y are true**
$x xor $y	**Xor**	**Returns true if $x is less than $y**

Increment/Decrement (Ternary) Operators

These operators, also known as ternary operators, increase/decrease a variable's value by one

EXAMPLE	NAME	RESULT
++$x	**Pre-increment**	**Adds one to x, then returns $x**
$x++	**Post-increment**	**Returns $x, then adds one to $x**
—$x	**Pre-decrement**	**subtracts one from x, then returns $x**
$x—	**Post-decrement**	**Returns $x, then subtracts one from $x**

String Operator

These are operators that are specially designed for strings:

EXAMPLE	NAME	RESULT
$str1 . $str2	**Concatenation**	**Returns the concatenation of $str1 and $str2**
$x$str1 .= $str2	**Concatenation assignment**	**Append $str2 to $str1**

Array Operators

These are operators that used to compare arrays (which we will discuss in the next hour):

EXAMPLE	OPERATOR	RESULT
$x + $y	**Union**	**Union of $x and $y**
$x == $y	**Equality**	**If $x and $y have the same value/key pairs, the result is true**
$x === $y	**Identity**	**If $x and $y have the same value/key pairs in the same order and of the same data type, the result is true**
$x != $y	**Inequality**	**If the value of $x is not equal to the value of $y, it returns true**
$x <> $y	**Inequality**	**If the value of $x is not equal to the value of $y, it returns true**
$x !== $y	**Non-indentity**	**If $x and $y are not identical, it returns true**

Summary

So far, we have covered a lot of things you need to know in order to get going and see PHP in action. We have installed a webserver package called XAMPP that comes with the APACHE web service that allows us to run PHP scripts in our browser. You should now be very knowledgeable enough to create and run PHP scripts and have a good idea on how to output information to the screen. We have also covered some of the basics of PHP like variables and constants, and looked at some of its operators; some of them will be useful in the next hour.

So let’s move on!

Chapter 2 - Hour 2: More PHP Basics

“Talk is cheap. Show me the code” – Thomas C. Gale

In this chapter we will cover:

- Conditional Statements
- Loops
- Arrays
- Functions

This is the part where we dive a little deeper into the basics of PHP and get to do a lot of cool stuff. You are done with salad portion of the basics, this is the meat section.

Conditional Statements

Conditional statements allow for a bit of decision making in code. They perform different actions depending on the whether the condition the programmer specified has been evaluated as true or false.

Here is a list of conditional statements used in PHP:

- *If* statements
- If…else statements
- If…else if…else statement
- Switch statements

Let us take a look at each one in detail.

The *if* Statement

An **if**statement evaluates an expression and executes a block of code if the expression evaluated to true. If it evaluated to false, the code block is ignored.

The following example will output “5 is less than 10!” if the condition is true, otherwise, the code block will be ignored:

```
<?php

    $x = 5;

    if ($x < 10) {

            echo “$x is less than 10!”;

    }
?>
```

Output:

5 is less than 10!

The *if…elseif* Statement

The **if…elseif** statement executes a code if block if the expression evaluated as true and another code if that expression is evaluated as false.

Let us alter the previous example a bit. The following code will output “50 is greater than 20!” which is on the second code block because it satisfies the second condition. The value assigned to **$x** is 50 and it will return false on the first condition:

```
<?php

    $x = 50;

    if ($x < 10) {

            echo “$x is less than 10!”;

    } else if ($x > 20) {

            echo “$x is greater than 20!”;

    }

?>
```

Output:

50 is greater than 20!

The *if…elseif…else* Statement

Same as the **if…elseif** statement. The difference is that when all other conditions fail, the last code block is executed.

In The example below will output “I5 is between 10 and 20!” because **$x** is 15. It needs to be less than 10 to satisfy the first condition, and greater than 20 to satisfy the second condition, otherwise, it is between 10 and 20:

```
<?php

    $x = 15;

    if ($x < 10) {

            echo “$x is less than 10!”;

    } else if ($x > 20) {

            echo “$x is greater than 20!”;

    }else {

            echo “$x is between 10 and 20!”;

    }

?>
```

Output:

I5 is between 10 and 20!

Switch Statement

A **switch** statement just matches the value of an expression with cases in the structure. If a match is found, the code block associated with that case is executed. When no match is found, the **default** statement is used. The **break** statement is what prevents the code from jumping into the next case automatically.

In the example below the value of **$lunch** will be checked against all the cases to *see* if any of them match. In this case, the output will be “Burgers are yummy!” since that is the corresponding case.

```
<?php

    $lunch = “burger”;

    switch ($lunch) {
            case “cereal”:
                    echo “Cereal? You had breakfast for lunch!”;
                    break;
            case “salad”:
                    echo “Salad makes for a healthy lunch!”;
                    break;
            case “burger”:
                    echo “Burgers are yummy!”;
                    break;
            default:
                    echo “looks like you skipped lunch. That is not good!”;
    }

?>
```

Output:

Burgers are yummy!

Loops

Here is a list of looping statements in PHP:

- while
- do…while
- for
- foreach

We shall go into each one in detail.

The While Loop

The **while** loop executes a block of code repeatedly as long as the specified condition remains true.

The code below shows the value **$x** being if it is less than or equal to 5. If the condition is true, it gets outputted to screen and then incremented again until the condition returns false:

```
<?php

    $x = 1;

    while ($x <= 5) {

            echo “The value of x is: ” . $x . “<br />”;
            $x++;

    }
?>
```

Output:

```
The value of x is: 1
        The value of x is: 2
        The value of x is: 3
        The value of x is: 4
        The value of x is: 5
```

Do…While Loop

Executes the block of code at least once then loops through it until the specified condition becomes false.

The example below shows the value of **$x** being outputted to the screen then subtracted by one at least once, then it is checked to see if it is greater than zero. If it is, then it repeats the process until it becomes less than zero:

```
<?php

    $x = 5;

    do {

            echo “The value of x is: ” . $x . “<br />”;
            $x—;

    } while ($x >= 0);

?>
```

Output:

The value of x is: 5
The value of x is: 4
The value of x is: 3
The value of x is: 2
The value of x is: 1
The value of x is: 0

For Loop

Loops through a code block a specified number of times.

There are three things needed in order for the loop to work. And the example below has them all:

1. **Initialization** – this is where a variable is declared to start the loop counter. This is**$x = 1**in our example. This mean that the **$x** is the initializer and it is set to start at 1

2. **Condition** – This is the second part of the loop where we check if the loop counter still matches a condition. When it returns false, the loop stops execution of the code block. Our condition in the example below is**$x <= 5**

3. **Increment** – When the loop ends, this part of the code is executed once and then the loop repeats. This is where we can do some operations to the loop counter. In our case we increase the value of**$x**by one with**$x++**

```
<?php

   for ($x = 1; $x <= 5; $x++) {

            echo “The value of x is: ” . $x . “<br />”;

   }

?>
```

Output:

```
The value of x is: 1
      The value of x is: 2
      The value of x is: 3
      The value of x is: 4
      The value of x is: 5
```

For each Loop

A loop that goes through each element found in an array. It executes a block of code with each element found in the array until it runs out.

The code below loops through all the elements of an array then outputs the value of the element to the screen:

```
<?php

    $foods = array(“cereal”, “salad”, “burgers”, “fries”);

    foreach ($foods as $food) {

            echo “$food <br />”;

    }

?>
```

Functions

A function is a piece of code that can be used of over and over in a program. A function takes one or more inputs called parameters (also called arguments) and does some operations to them. A function can sometimes return a value after it is done processing.

A few things to remember about functions:

- They are not case-sensitive
- They are not executed automatically when the page loads until they are called by a certain piece of code
- PHP has more than 1000 built in functions that can cater to your requirements

Creating a Function

Creating your own function is easy; just start with the keyword **function** followed by the function name and parenthesis. After that place your code between two curly braces { and } called the opening and closing braces, respectively. Now you can use the function as many times as you need it. This saves you the trouble of having to re-type the code every time.

The code below demonstrates a function that outputs a message to the screen

```
<?php

    functionechoMessage() {

            echo “Hello, I am a PHP function!”;

    }

    echoMessage();
?>
```

Output:

Hello, I am a PHP function!

Parameters for Functions

Sometimes you can pass information to functions so that operations are done on them. This information is known as parameters and is placed in the parenthesis. You can add multiple parameters by separating them with a coma.

The below example shows a function that subtracts two numbers supplied to function as parameters:

```
<?php

   functionsubFunction($num1, $num2) {

            $diff = $num1 - $num2;
            echo “The difference between $num1 and $num2 is: ” . $diff;

   }

   subFunction(5, 6);

?>
```

Output:

The difference between 5 and 6 is: -1

Function that Return a Value

A function can not only accept information, but they can also give it back in form of a return value. Just write the **return** keyword followed by the value or object you wish the function to return. This stops execution of the function.

In the following example we use the function to add to numbers and then return the sum. We then assign the sum of numbers to the variable **$return_val** then output that to the screen.

```
<?php

    functionaddFunction($num1, $num2) {

            $sum = $num1 + $num2;
            return $sum;

    }
    $return_val = addFunction(20, 15);
    echo “The value returned by the function is: ” . $return_val;

?>
```

Output:

The value returned by this function is: 5

Default Values for Parameters

In case a function that needs parameters is called without any parameters, we can specify a default value for that parameter instead.

In the example below, we call a function three times. We don't pass a value in the first function call just so we demonstrate the difference:

```
<?php

    functionPrintName($name = "Jack") {

            echo "Hi, my name is $name <br />";

    }

    PrintName(); // will use the default value of Jack
    PrintName("Jill");
    PrintName("Adam");

?>
```

Output:

```
    Hi, my name is Jack
        Hi, my name is Jill
        Hi, my name is Adam
```

Arrays

An array is data structure that can store multiple homogenous values in a single variable. Each value is associated with a key that is used to identify it and return it.

Numeric Array

These are arrays that use numbers as their index. The index begins at zero by default.

They are two ways to create numeric arrays:

1. **This way automatically assigns the index:**

   $dairy = array(“yoghurt”, “ice cream”, “cheese cake”);

2. **This way assigns the index manually:**

   $dairy[0] = “yoghurt”
   $dairy[1] = “ice cream”
   $dairy[2] = “cheese cake”

The example below shows how to get values from a numeric array using indices:

```
<?php
    $dairy = array("yoghurt", "ice cream", "cheese cake");

    // accessing values by their index
    echo "I to eat " . $dairy[0] . ", " . $dairy[1] . " and " . $dairy[2]. "!";

    $arrayCount = count($dairy); // PHP built in function that returns the number of elements in the array

    // looping through a numeric array using a for loop
    for ($i = 0; $i < $arrayCount; $i++) {

            echo $dairy[$i] . "<br />";

    }

?>
```

Output:

```
I to eat yoghurt, ice cream and cheese cake!
        yoghurt
        ice cream
        cheese cake
```

Associative Array

Use strings as their index that make for a strong association between key and value.

They are two ways to create associative arrays. We will demonstrate by using prices of fantasy suits:

1. **Firsts method**:

   $suit_prices = array(“batman” => 4000, “arrow” => 1500, “superman” => 0);

2. **Second method**:

$suit_prices[“batman”] => 4000;

$suit_prices[“arrow”] => 1500;

$suit_prices[“superman”] => 0;

Note: When outputting values of associative arrays, don’t put the associative array inside double quotes, otherwise, the values won’t get outputted.

The example below demonstrates how to access value for an associative array:

```
<?php
   $suit_prices = array("batman" => 4000, "arrow" => 1500, "superman" => 0);

   echo "The Bat suit costs: " . $suit_prices['batman'] . "<br />";
   echo "The Green Arrow suit costs: " . $suit_prices['arrow'] . "<br />";
   echo "The Superman is free. It suit costs: " . $suit_prices['superman'] . "<br />";

?>
```

Output:

```
The Bat suit costs: 4000
      The Green Arrow suit costs: 1500
      The Superman suit is free. It costs: 0
```

The other type of array worth mentioning is a multidimensional array These are arrays that contain other arrays. But they are an advanced topic that you can look into further.

Summary

Congratulations! You are now acquainted with the basics of PHP. By now you should be able see the power PHP offers you. In this chapter we discussed conditional statements that allow the code you write to make decisions based on specified conditions. Then we looked at loops that execute a block of code until a condition specified return false or until they run out elements. Next we discussed how to call code multiple times in a program by putting them in a function, and also how to pass them information and have them return values to us. Finally, we looked into numeric and associative arrays. We created them and returned values by either accessing them directly or looping through them.

Now you are ready to take a breather from PHP and make another cup of coffee. This only gets more interesting!

Chapter 3 - Hour 3: MySQL Basics

“Tell me and I forget, teach me and I may remember, involve me and I learn.” - Benjamin Franklin

In this chapter we will cover:

- **Start MySQL**
- **Create and Deleting Databases**
- **Creating, Reading, Updating and Deleting Tables (CRUD)**

Now that we have looked into PHP, let’s look at what enables the PHP to populate webpages with content. That would be a database such as MySQL. We will look at most of the things you need to know in order to consider yourself a master of the basics of MySQL and do what you need to for that next step in achieving your dreams.

This will be a breeze as we will learn how to do some really cool things with databases.

Below is a picture of the table we will create and be working with in the database. We will just be adding albums to a music database. You have the table name at the very top and the fields or columns below the table name.


Albums
AlbumID
AlbumName
AlbumArtist
AlbumType
NumberOfTracks
DateReleased

Starting MySQL Client

What we are to be doing for this hour is entering MySQL commands using the command-line of the MySQL client that comes with XAMPP by opening up Windows command prompt and navigating to the root folder of where XAMPP is installed.

When you open command prompt, enter the below command to navigate to the “C” drive, Assuming XAMPP is installed there, and then press enter after that:

**cd **

Now, navigate to where the executable for the client is installed to start it. That is in *xampp->mysql->bin* by entering the following command and then pressing enter:

cdxampp\mysql\bin

Once there, type the following command:

mysql -u your_username -p

Replace “**your_username**” with your actual username. This is “root” by default. You will then be asked to enter your password. Once you do that, press enter. The password is blank by default and you really ought to change that for security reasons. But that is beyond the scope of this book.

If you notice with command-line is being prompted by something called MariaDB. Don’t get confused because it is just a community-developed fork of the MySQL relational database management system. A large number of SQL commands will still work and unlike the previous MySQL prompt, it lets you known which database you have selected.

Now we are ready to get started!

TIP: Alternatively you can use the phpmyadmin interface to create databases and table with just a few clicks. You can also type SQL commands there but since we want to get hard-core as fast as possible, we won’t be doing that. Besides, nothing in this hour has been hard so far, I hope.

Creating and Deleting a Database

Let’s start by creating a database by entering the following command:

```
CREATE DATABASE music;
```

Always remember this:

- From here onwards, all statements and commands typed should end with semicolon or **/g**. Otherwise the client will think you are still typing the command and it won’t execute
- Although not required, because command line is not case sensitive, MySQL commands are typed in uppercase letters to distinguish between commands and things like database names, table names, columns and rows

To delete the database it is as easy as entering:

```
DROP database_name;
```

Where “database_name” is the name of the database you want to drop.

Now that we have created our database, we need to select it in order to create tables. We type the following into you command line to select the database we just created, and then press enter:

USE music;

WARNING: Don’t forget to select the database otherwise you will get errors when you try to create tables etc.

NOTE: the command-line prompt will let you know which database you have selected. It should look something like this “Maria DB [music]>”.

Creating, Altering and Deleting Tables

Now that we have selected out database, let’s create some tables. The first table we will create is about information on the artist.

It is good practice (and headache saving) to first check if any tables with the same name exists because you can’t create two tables with the same name in the same database. If one is found, it is deleted along with its data. So, be careful with this command especially if the table is related to other tables.

Creating Tables

We create the table as follows:

```
CREATE TABLE Album
(
        AlbumIDsmallint unsigned NOT NULL auto_increment PRIMARY KEY,
        AlbumNamevarchar(255) NOT NULL,
        AlbumArtistvarchar(80) NOT NULL,
        AlbumType ENUM('A', 'E', 'M') NOT NULL,
        numTracksvarchar(3) NOT NULL,
        DateReleasedDATE NOT NULL
);
```

TIP: When typing large commands it is easy to mess up. Especially when using thecommand line. If this happens just press the up arrow on your keyboard to bring up previous commands and use the left and right arrows to navigate to where you messed up and fix.

To see the list of tables in you database, enter the following piece of code:

```
SHOW tables;
```

You see the following output:

```
+————————+
| Tables_in_music |
+————————+
| album          |
+————————+
```

When you press Enter, the following table will be added to the music database. Let us break down what is happening:

1. **Creating the Album Table**

   CREATE TABLE Artists () creates a new table and the lines inside the parenthesis define the table's structure.

2. **Giving Each Album a Unique ID**

   The field (or column) AlbumsID has the following properties:

   - **smallint unsigned** (unsigned small integer) – this means we can store artists in the range 32,278 to – 32,767**.**We will look at the common data types that you will be working with later on in the chapter.
   - **NOT NULL** – this means the field cannot be left blank when creating a record.
   - **auto_increment –** this just means that you don't have to enter anything in the field. When a record is created, it will be assigned a new value by MySQL.
   - **PRIMARY KEY** -Each table can only have one **Primary Key.** It is used to uniquely identify a record/row in a database and, once created, it can never be changed. This makes it quicker to find (at the expense of some storage space).

3. **Adding the AlbumName Field**

   Next we create a field to store the name of the album. The data type **varchar**stands for variable number of character and stores a string of up to 225 characters.

4. **Adding the AlbumArtist Field**

   This field also stores a variable number of characters and its string is limited to only 80 characters.

5. **Adding the AlbumType Enumerated Type Field**

   The enumerated type **ENUM** is a string object that has set of values that are allowed. In our case, the allowed values are '**L**' for Long Play (LP), '**E**' for Extended Play (EP) and '**M**' for a mix tape.

6. **Adding the numTracks Field**

   If this seems wrong to you, then you have been paying attention! This field is used to store the number of tracks the album has and should be a numeric type and not a string type. This was done intentionally to demonstrate how to change the data type and name of a field/column later of in the chapter.

7. **Adding DateReleased Field**

   The last line created stores the date the album was released. The date data type is used to store date values.

Let us look at some of the common data types that you will be using:

- **Numeric Types**
  1. **INT:** whole number 2^31 to -2^31 – 1
  2. **FLOAT:** Decimal spaces 1.1E38 to -1.1E37
  3. **Double:** Decimal spaces 1.7E308 to -1.7E307
- **String Types**
  1. **CHAR:** fixed length character string
  2. **VARCHAR:** Character string with a variable length
  3. **ENUM:** A character string that has a limited number of total values which you must define
- **Date and Time**
  1. **DATE:** YYYY:MM:DD
  2. **TIME:** HH:MM:SS
  3. **DATETIME:**YYYY-MM-DD HH:MM:SS
  4. **YEAR:** YYYY

Altering Tables

Sometime you can make a mistake or may just wish to changes things in your table. Luckily, you can do so with MySQL by altering the table using MySQL statements. We will look at changing table and column names, and changing data types.

First we need to look at the table we just created by the typing following command to describe our table. We should be able to see its structure in the command line:

DESCRIBE Album;

You should see the following output:

```
+————+————————+—+———+———+————+
| Field      | Type               | Null | Key | Default | Extra          |
+————+————————+—+———+———+————+
| AlbumID    | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| AlbumName  | varchar(255)       | NO   |     | NULL    |                |
| AlbumArtist | varchar(80)       | NO   |     | NULL    |                |
| AlbumType  | enum('A','E','M')  | NO   |     | NULL    |                |
| numTracks  | varchar(3)         | NO   |     | NULL    |                |
| DateReleased | date             | NO   |     | NULL    |                |
+————+————————+—+———+———+————+
```

Looks like we can indeed change a few things:

1. **Changing the Table Name**

   Seems the singular form "Album" doesn't make much sense as a table is used to store multiple records. We should use the plural form and change it from Album to Albums. Just so it makes more sense.

   Here is how we change the table name:

   ```
   RENAME TABLE Album to Albums;
   ```

   If we enter the show tables command we should see the table name has changed:

```
+————————+
| Tables_in_music |
+————————+
| albums          |
+————————+
```

2. **Changing The Data Type of numTracks**

   The field **numTracks** should not be a string type but numeric. The following code should change its data type to the appropriate one:

```
ALTER TABLE Albums
    MODIFY COLUMN numTrackssmallint(2) NOT NULL;
```

3. **Changing the Column Name for numTracks**

   It seems we are still not done with the field **numTracks**. We need to change it to make it a little more readable by changing it from **numTracks** to **NumberOfTracks.**

```
ALTER TABLE Albums
    CHANGE numTracksNumberOfTrackssmallint(2) NOT NULL;
```

   NOTE: we still have to define the data type even when changing the name of the column

With the following changes, when we enter the statement to describe the table, we should see the following:

```
+———-+————-+—+—+—+———+
| Field        | Type              | Null | Key | Default | Extra         |
+———-+————-+—+—+—+———+
| AlbumID      | smallint(5) unsigned | NO   | PRI | NULL    | auto_increment |
| AlbumName    | varchar(255)      | NO   |     | NULL    |               |
| AlbumArtist  | varchar(80)       | NO   |     | NULL    |               |
| AlbumType    | enum('A','E','M')  | NO   |     | NULL    |               |
| NumberOfTracks | smallint(2)     | NO   |     | NULL    |               |
| DateReleased | date              | NO   |     | NULL    |               |
+———-+————-+—+—+—+———+
```

That looks much better!

Deleting Tables

The code below can be used to delete a table and all its data from the database:

```
DROP TABLE table_name;
```

Where table_name is the name of the table you wish to delete.

Inserting Records into a Table

We use the INSERT INTO statement in order to insert new records in a table.

INSERT INTO Albums VALUES

(NULL, “Rush of Blood to the Head”, “Coldplays”, ‘A’, 12, “2002-08-02”),

(NULL, “Dirty Gold”, “Angel Haze”, ‘A’, 16, “2013-12-30”),

(NULL, “Back to the Woods”, “Angel Haze”, ‘M’, 13, “15-09-15”),

(NULL, “Parachutes”, “Coldplay”, ‘A’, 10, “1999-07-10”),

(NULL, “1989 (Delux Edition)”, “Taylor Swift”, ‘A’, 15, “2014-10-27”),

(NULL, “Prospekts March”, “Coldplay”, ‘E’, 8, “2008-09-21”),

(NULL, “Random Access Memories”, “Daft Punk”, ‘A’, 14, “2013-05-13”),

(NULL, “X & Y”, “Coldplay”, ‘A’, 13, “2005-06-01”);

Seems we like Coldplay a bit too much. Who doesn’t?

The SELECT Statement

The **SELECT** statement allows you to retrieve data from a data base. The following example demonstrates how to use **SELECT** statement to retrieve all the records from the Albums table:

```
SELECT * FROM Albums;
```

The output should be something like this:

```
+—-+————+———+———+———-+———+
| AlbumID | AlbumName              | AlbumArtist  |AlbumType | NumberOfTracks | DateReleased |
+—-+————+———+———+———-+———+
|      1 | Rush of Blood to the Head | Coldplays    | A        |           12 | 2002-08-02   |
|      2 | Dirty Gold              | Angel Haze   | A        |           16 | 2013-12-30   |
|      3 | Back to the Woods       | Angel Haze   | M        |           13 | 2015-09-15   |
|      4 | Parachutes              | Coldplay     | A        |           10 | 1999-07-10   |
|      5 | 1989 (Delux Edition)    | Taylor Swift | A        |           15 | 2014-10-27   |
|      6 | Prospekts March         | Coldplay     | E        |           8 | 2008-09-21   |
|      7 | Random Access Memories  | Daft Punk    | A        |           14 | 2013-05-13   |
|      8 | X & Y                   | Coldplay     | A        |           13 | 2005-06-01   |
+—-+————+———+———+———-+———+
```

To select specific columns we use the following **SELECT** statement:

```
SELECT AlbumName, AlbumArtist, DateReleased, NumberOfTracks
FROM Albums;
```

The output should be:

```
+————————+———+————+———-+
| AlbumName              | AlbumArtist  |DateReleased | NumberOfTracks |
+————————+———+————+———-+
| Rush of Blood to the Head | Coldplays    | 2002-08-02   |           12 |
| Dirty Gold              | Angel Haze   | 2013-12-30   |           16 |
| Back to the Woods       | Angel Haze   | 2015-09-15   |           13 |
| Parachutes              | Coldplay     | 1999-07-10   |           10 |
| 1989 (Delux Edition)    | Taylor Swift | 2014-10-27   |           15 |
| Prospekts March         | Coldplay     | 2008-09-21   |           8 |
| Random Access Memories  | Daft Punk    | 2013-05-13   |           14 |
| X & Y                   | Coldplay     | 2005-06-01   |           13 |
```

+------------+----------+----------+---------+

The WHERE Clause

Sometimes you will need to filter out records and select only those records that fulfil a certain criterion. This is where the **WHERE** clause comes into play as the following example demonstrates:

```
SELECT AlbumName, AlbumArtist, DateReleased, NumberOfTracks
FROM Albums
WHERE AlbumArtist = 'Coldplay'
```

This should output:

```
+————————+———-+————————+————-+
| AlbumName       | AlbumArtist | DateReleased | NumberOfTracks |
+————————+———-+————————+————-+
| Parachutes      | Coldplay    | 1999-07-10   |             10 |
| Prospekts March | Coldplay    | 2008-09-21   |              8 |
| X & Y           | Coldplay    | 2005-06-01   |             13 |
+————————+———-+————————+————-+
```

NOTE: *This example was done on a text field, hence, the single quotes on the value we want to check against. You can check against numeric fields too. Just remove the single quotes and make sure you type a numeric value that corresponds to the numeric data type of that field.*

You can further filter out the record using the **AND**and**OR** operators to narrow down the results further:

The **AND** operator displays records that match both conditions of the criterion:

```
SELECT AlbumName, AlbumArtist,AlbumType
FROM Albums
WHERE AlbumArtist = 'Coldplay' AND AlbumType = 'A';
```

The output should display results where the **AlbumName** is Coldplay and the **AlbumType** is 'L'. If any record doesn't match that criterion, it will be excluded:

```
+——+——-+——+
| AlbumName  |AlbumArtist | AlbumType |
+——+——-+——+
| Parachutes | Coldplay    | A        |
| X & Y      | Coldplay    | A        |
+——+——-+——+
```

The **OR** operator is used to return records that match either of the one of the criterion specified.

```
SELECT AlbumName, AlbumArtist, AlbumType
FROM Albums
WHERE AlbumType = 'E' OR AlbumType = 'M';
```

In the above example, any record that has either **AlbumType** of 'E' or **AlbumType** of 'M' will be displayed. The rest will be excluded. The output should be as follows:

```
+———-+——-+——+
| AlbumName        | AlbumArtist | AlbumType |
+———-+——-+——+
| Back to the Woods | Angel Haze  | M        |
| Prospekts March   | Coldplay    | E        |
+———-+——-+——+
```

The ORDER BY Keyword

In order to sort the **result_set** (the table returned by a **SELECT** statement) by one or more columns, we use the **ORDER BY** keyword and specify whether it should be in descending or ascending order:

```
SELECT AlbumName, AlbumArtist
FROM Albums
ORDER BY AlbumArtist ASC;
```

NOTE: By default, they are sorted in ascending order. This means that if we leave out the ASC keyword, it would not make a difference. We can sort them in descending order by replacing ASC with DESC in the above example.

That should output records in ascending order based on the name of the artist of the album:

```
+———————+———+
| AlbumName           | AlbumArtist  |
+———————+———+
| Dirty Gold          | Angel Haze   |
| Back to the Woods   | Angel Haze   |
| Parachutes          | Coldplay     |
| Prospekts March     | Coldplay     |
| X & Y               | Coldplay     |
| Rush of Blood to the Head | Coldplays    |
| Random Access Memories    | Daft Punk    |
| 1989 (Delux Edition)      | Taylor Swift |
+———————+———+
```

The UPDATE Statement

Looking back at the some of the records we have displayed so far, it seems we made a mistake and named Coldplay as Coldplays in one of the records. This can be corrected by updating that record using the **UPDATE** statement..

If we output the record now with the following code:

```
SELECT AlbumID, AlbumName, AlbumArtist
FROM Albums;
```

The output should be:

```
+—-+————-+———+
| AlbumID | AlbumName              | AlbumArtist  |
+—-+————-+———+
|      1 | Rush of Blood to the Head | Coldplays    |
|      2 | Dirty Gold             | Angel Haze   |
|      3 | Back to the Woods      | Angel Haze   |
|      4 | Parachutes             | Coldplay     |
|      5 | 1989 (Delux Edition)   | Taylor Swift |
|      6 | Prospekts March        | Coldplay     |
|      7 | Random Access Memories | Daft Punk    |
|      8 | X & Y                  | Coldplay     |
+—-+————-+———+
```

We update that recode like this:

```
UPDATE Albums
SET AlbumArtist = ‘Coldplay’
WHERE AlbumID = 1;
```

Warning: Don’t omit the WHERE clause or all the records in the table will be updated.

If we output it now, we should see the record has been updated:

```
+——+————————+———+
| AlbumID | AlbumName              | AlbumArtist  |
+——+————————+———+
|      1 | Rush of Blood to the Head | Coldplay     |
|      2 | Dirty Gold             | Angel Haze   |
|      3 | Back to the Woods      | Angel Haze   |
|      4 | Parachutes             | Coldplay     |
|      5 | 1989 (Delux Edition)   | Taylor Swift |
|      6 | Prospekts March        | Coldplay     |
|      7 | Random Access Memories | Daft Punk    |
|      8 | X & Y                  | Coldplay     |
+——+————————+———+
```

Deleting Records in a Tables

To delete records in a table, we use the **DELETE** statement. The following example deletes the record/row from the table with an **AlbumID** of 1:

```
DELETE FROM Albums
WHERE AlbumID = 1;
```

Warning: Don't omit the WHERE clause or all the records in the table will be deleted.

Now we output the records in the table. We will limit the number of columns to only **AlbumID, AlbumName** and **AlbumArtist.** The record with an **AlbumID** of 1 should be missing from the result_set because it has been deleted.

```
+—-+————+———+
| AlbumID | AlbumName          | AlbumArtist  |
+—-+————+———+
|      2 | Dirty Gold          | Angel Haze   |
|      3 | Back to the Woods   | Angel Haze   |
|      4 | Parachutes          | Coldplay     |
|      5 | 1989 (Delux Edition)   | Taylor Swift |
|      6 | Prospekts March     | Coldplay     |
|      7 | Random Access Memories | Daft Punk    |
|      8 | X & Y               | Coldplay     |
+—-+————+———+
```

If you want to delete all records in the table, the following DELETE statement will do just that:

```
DELETE from Albums;
```

Summary

Okay, now that you are familiar with basics of MySQL and can access it using the command line and use it to create and delete databases; create, alter and delete tables; select, insert, update, and delete table data; and do other neat things with data like filtering it and ordering it the way you want using MySQL commands, it is the time to move on and look at how we can use PHP and MySQL together and create a very simple website where you add records to a database of our favorite artists.

Chapter 4 - Hour 4: PHP and MySQL: The Dynamic Duo

“A language that doesn’t affect the way you think is not worth knowing” – Alan J. Perils

In this chapter we will cover:

- Creating the Family Database
- Creating and Inserting Values into the Family Members Table
- Creating a PHP configuration File the Website
- Creating the Index Page
- Creating the Form to Input Data
- Creating the Member Added Page

Think of seeing your two favorite artists performing together. This is what it should feel like at this movement as we are about see PHP and MySQL (mostly PHP) work together to create a website. By now you should be able to imagine the good music these two can make together.

The website is simple; let’s say you want to keep track of your family members online. We will create a simple website that lets you add them (if you like them) to a database so you don’t always have to remember who they are. We will forego a lot of security features just to keep it simple and other functions but this simple example should be enough to get you on your way.

By now you have a fairly good grasp of PHP and can use the command line to type SQL commands. We will need both. So make sure both Apache and MySQL are running in XAMPP (See Chapter 1 and Chapter 3 on how to enable Apache and MySQL respectively).

So let us get started!

Creating Family Database

Open up the command line and fire up the MySQL Client and login as the ‘root’ (see Chapter 3).

Create the Database by entering the following code:

```
CREATE DATABASE Family;
```

Then select the database in order to use it:

```
USE Family;
```

Create and Inserting Values into the Family Members Table

Now we create the table that stores the information. It should look like this:

Family Members
Member ID
First Name
Last Name
Age
Gender
Relationship

The code to create the tables looks as follows:

```
CREATE TABLE FamilyMembers
(
    MemberIDint unsigned NOT NULL auto_increment PRIMARY KEY,
    FirstNamevarchar(80) NOT NULL,
    LastNamevarchar(80) NOT NULL,
    Age int(3) unsigned NOT NULL,
    Gender ENUM (‘Male’, ‘Female’) NOT NULL,
    Relationship varchar(80) NOT NULL
);
```

When we describe the table we get something like this:

```
+————+——————+—+——+——+————-+
| Field       | Type                | Null | Key | Default | Extra          |
+————+——————+—+——+——+————-+
| MemberID    | int(10) unsigned    | NO   | PRI | NULL    | auto_increment |
| FirstName   | varchar(80)         | NO   |     | NULL    |                |
| LastName    | varchar(80)         | NO   |     | NULL    |                |
| Age         | int(3) unsigned     | NO   |     | NULL    |                |
| Gender      | enum('Male','Female') | NO   |     | NULL    |                |
| Relationship | varchar(80)        | NO   |     | NULL    |                |
+————+——————+—+——+——+————-+
```

Then we insert values into the tables:

```
INSERT INTO FamilyMembers VALUES
(NULL, 'Robert', 'Williams', 52, 'Male', 'Father'),
(NULL, 'Martha', 'Williams', 49, 'Female', 'Mother'),
(NULL, 'Doreen', 'Williams', 18, 'Female', 'Sister'),
(NULL, 'Jace', 'Williams', 13, 'Female', 'Adoptive Sister'),
(NULL, 'Benson', 'Williams', 22, 'Male', 'Brother'),
(NULL, 'Timothy', 'James', 23, 'Male', 'Cousin');
```

When we select all the records in the table, we see something like this:

```
+——-+———+——-+——+———+————+
| MemberID | FirstName | LastName | Age | Gender | Relationship    |
+——-+———+——-+——+———+————+
|       1 | Robert    | Williams |  52 | Male   | Father         |
|       2 | Martha    | Williams |  49 | Female | Mother         |
|       3 | Doreen    | Williams |  18 | Female | Sister         |
|       4 | Jace      | Williams |  13 | Female | Adoptive Sister |
|       5 | Benson    | Williams |  22 | Male   | Brother        |
|       6 | Timothy   | James    |  23 | Male   | Cousin         |
+——-+———+——-+——+———+————+
```

Now we are done with the command line (but not MySQL), so we can close that.

Create PHP configuration File for the Website

We must first create a folder and name it “Family” and place in it in the htdocs folder. This folder is located in the root directory of our XAMPP installation. Then we create a file called config.php and place it in the root folder of our website

Warning: *for security purpose this file needs to be placed outside of the server because, while it is not easy to read PHP through the browser, it can happen if the server is misconfigured. But since we want to keep this as simple as possible, we will place it here.*

Next we define the constants needed to establish out connection string that we shall use too connect to the music database:

```
<?php
        define('DB_NAME', 'Family');
        define('DB_USER', 'root');
        define('DB_PASSWORD', '');
        define('DB_HOST', 'localhost');
?>
```

Warning: *for security reason, putting in a plain text password is a major security risk. You can use hash() and store the hash in the configuration file config.php. Then you can check if the entered password matches the hash when logging in.*

Then we establish the connection string by passing in the constants; host name, username, password, and database name to the method that establishes the connection:

```
<?php
    define('DB_NAME','Family');
    define('DB_USER','root');
    define('DB_PASSWORD','');
    define('DB_HOST','localhost');
    $conn = @mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

?>
```

Now we need to check if the connections succeeded or not. If it didn't, we display the appropriate error message and stop execution of the PHP script:

```
<?php
    define('DB_NAME','Family');
    define('DB_USER','root');
    define('DB_PASSWORD','');
    define('DB_HOST','localhost');
    $conn = @mysqli_connect(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);
    if (!$conn) {
            die('Could not connect: ' . mysqli_connect_error());
    }

?>
```

And that's it for establishing the connection string. This script will be required by every PHP script that needs to connect to the database to work with the data.

Creating the Index Page

Now we need to create the landing page which shall list the family members that are in our family database in a table. We are going to add some CSS (by linking to external style sheets) to create an attractive table to display our data in.

We create a file named index.php and place it root directory of the website. The code for file should look something like this:

```
<!DOCTYPE html>
<html>

<head>
<link rel="stylesheet" type="text/css" href="css/main.css">
<link rel="stylesheet" type="text/css" href="css/table_styling.css">
</head>

<body>

<div id="container">

<div id="header">
<h2>My Family</h2>
</div>

<div id="btn_add">
<a href="form.php">ADD</a>
</div>

<div id="content">
<?php
```

```
require_once('config.php');

                    $sql = "SELECT MemberID, FirstName, LastName, Age,
Gender, Relationship
                                FROM FamilyMembers";

   $response = @mysqli_query($conn, $sql);

                    if ($response){

echo '<table>
                                <tr>
                                        <th>Member ID</th>
                                        <th>First Name</th>
                                        <th>Last Name</th>
                                        <th>Age</th>
                                        <th>Gender</th>
                                        <th>Relationship</th>
                                </tr>';

                            while ($row = mysqli_fetch_array($response)){

                                echo '<tr>
                                        <td>' . $row['MemberID'] .
'</td>
                                        <td>' . $row['FirstName'] .
'</td>
                                        <td>' . $row['LastName'] .
'</td>
                                        <td>' . $row['Age'] . '</td>
                                        <td>' . $row['Gender'] . '</td>
                                        <td>' . $row['Relationship'] .
'</td>
```

```
                                        </tr>';

                                    }

                                    echo '</table>';

                                }
                                else {

                                    echo "Cound not get a response from database " .
mysqli_error($conn);

                                }

                                mysqli_close($conn);

                            ?>

</div>

</div>

</body>

</html>
```

There is quite a lot going on here but we will not get into it all. We also won't be discussing any of the HTML and CSS as this book assuming you have some basic knowledge of those and how to link external style sheets.

Firstly, in the PHP code, we include the **config.php** file by calling the **require_once** function and passing the path to the file as a parameter (we pass in a relative path since the configuration file). This will allow us to access our database connection we defined in that file:

require_once('config.php');

Next we create the query string that will be used to retrieve data from database using SQL and assign it to the variable**$sql**:

$sql = "SELECT MemberID, FirstName, LastName, Age, Gender, Relationship FROM FamilyMembers";

We use a**SELECT**statement to select the fields we wish to display from the **FamilyMembers**table following the same syntax rules we would use on the command line.

Then we send the query to the database using the **mysqli_query** function and pass in the connection variable, which is now accessible because we included the config.php, and the query string then assign the return value to the $response variable:

$response = @mysqli_query($conn, $sql);

After that, we check if we got a response from the database using an **if** statement:

if ($response){

}

If we got a response from the database, we create a with a table row and table headers by embedding HTML in PHP code and echoing that inside the **if** statement:

```
echo '<table>
    <tr>
            <th>Member ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
            <th>Gender</th>
            <th>Relationship</th>
    </tr>';
```

Okay, that does it for the columns. Now we need to display the rows one by one dynamically in case we have many. The following code does that:

```
while ($row = mysqli_fetch_array($response)){

    echo ‘<tr>
            <td>’ . $row[‘MemberID’] . ‘</td>
            <td>’ . $row[‘FirstName’] . ‘</td>
            <td>’ . $row[‘LastName’] . ‘</td>
            <td>’ . $row[‘Age’] . ‘</td>
            <td>’ . $row[‘Gender’] . ‘</td>
            <td>’ . $row[‘Relationship’] . ‘</td>
    </tr>’;

}
```

In the above code, we use a loop to go through an array we get from the response using the **mysqli_fetch_array()**function and passing in the **$response** variable as a parameter. We assign the value (which is also an array contain the fields of the row) we get from the function into the variable **$row** and embed HTML around them to create new table rows dynamically as many as they are elements in the array.

Then we close of the table off and that is it for displaying the data dynamically:

```
echo ‘</table>’;
```

But what if there was no response for the database because something went wrong? That could happen. In that case we add an ***else*** statement after our ***if*** statement to display the error message using the **mysqli_error** function and we pass in our connection as a parameter:

```
else{

    echo “Cound not get a response from database ” .          mysqli_error($conn);

}
```

Lastly, we close the database connection using the **mysqli_close** function, passing in the connection as a parameter and we are done with the index page:

```
mysqli_close($conn);
```

Go into the browser and enter that path to the index.php as follows then press Enter:

https://localhost/Family/index.php

The below window should display:

As you can see; it is displaying the records we entered previous and has generated the

table dynamically. Neat huh!

CSS for index Page

Create a folder inside the websites root folder called css and create two CSS files called main.css and table_styling.css. These are the external style sheets for the page.

We enter the following code into main.css to style the body, headers, and links. It also centers the content:

```
body {
background-color: #EEE;
}

#container {
width: 900px;
margin-left: auto;
margin-right: auto;
}

#header {
color: #2ca089;
text-align: center;
font-size: 2em;
}

/* links */

a:link, a:visited {
background-color: #2ca089;
color: white;
padding: 5px 12px;
text-align: center;
```

```
text-decoration: none;
display: inline-block;
}

a:hover, a:active {
background-color: #AFDCB1;
}

#btn_add {
padding: 5px 0px;
text-align: right;
}
```

Then we enter the following code into table_styling.css to style the table:

```
table {
border-collapse: collapse;
width: 100%;
}

th, td {
text-align: left;
padding: 8px;
}

tr:nth-child(even){background-color: white; }

th {
background-color: #2ca089;
color: white;
}
```

Creating the Form to Input Data

We create a file called form.php and place it the root folder of the website. Then we place the code below:

```
<!DOCTYPE html>
<html>

<head>
<link rel="stylesheet" type="text/css" href="css/main.css">
<link rel="stylesheet" type="text/css" href="css/form_styling.css">
</head>

<body>

<div id="container">

<div id=content>

<div>
<form action="member_added.php" method="post">

<div id="header">
<h3>Add Family Member</h3>
</div>

<div id="btn_add">
<a href="index.php">View Members</a>
</div>

<label for="fname">First Name</label>
```

<input type=“text” id=“first_name” name=“first_name” maxlength=“80”>

<label for=“lname”>Last Name</label>
<input type=“text” id=“last_name” name=“last_name” maxlength=“80”>

<label for=“age”>Age</label>
<input type=“number” id=“age” name=“age” min=“1” max=“100”>

<label for=“gender”>Gender</label>
<select id=“gender” name=“gender”>
<option value=“Male”>Male</option>
<option value=“Female”>Female</option>
</select>

<label for=“relationship”>Relationship</label>
<input type=“text” id=“relationship” name=“relationship” maxlength=“80”>

<input type=“submit” name=“submit” value=“ADD”>
</form>
** **

** **

** **

</body>

</html>

Nothing out of the ordinary is happening here. This just your basic HTML form which we will use to send the data inputted into the fields to the database. The form has the two attributes which are of interest to us:

form action - This tells the form to send the form-data to the member_added.php script(which we shall create in a moment) that will process the information when the ADD button is pressed

method = "post" – Tells the form to send the form-data as an HTTP post transaction. This means that the data is appended inside the body of the HTTP request URL. This is good for sending sensitive data (things like passwords) unlike the "get" method which just appends the data to the URL and anyone one can see it.

Go into the browser and display the index page. Click the ADD button at top right of the table and the following page should display with the form to add family members:

CSS for Form Page

Create a CSS file called form_stlying.php and place it in the root directory of the website. Place the following style rules in that file to style the form using this external style sheet:

```
input[type=text], input[type=number], select {
    width: 100%;
    padding: 12px 20px;
    margin: 8px 0;
    display: inline-block;
    border: 1px solid #ccc;
    border-radius: 4px;
    box-sizing: border-box;
}

input[type=submit] {
    width: 100%;
    background-color: #2ca089;
    color: white;
    padding: 14px 20px;
    margin: 8px 0;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

input[type=submit]:hover {
background-color: #2ca089;
}
```

Creating the Member Added Page

Finally, you need to create one final PHP script for to process the data sent by the form. We create a new PHP script and called member_added and place in the root directory of our website. Then we place the following code in the script:

```
<!DOCTYPE html>
<html>

<head>
<link rel="stylesheet" type="text/css" href="css/main.css">
<link rel="stylesheet" type="text/css" href="css/form_styling.css">
</head>

<body>

<div id="container">

<div id="content">

<?php

require_once('config.php');

if(isset($_POST['submit'])) {

    $null_fields = array();

if (empty($_POST['first_name'])) {

    $null_fields[] = 'First Name';
```

```
    } else {

    $first_name = trim($_POST[‘first_name’]);
    }

if (empty($_POST[‘last_name’])) {

    $null_fields[] = ‘Last Name’;

    } else {

    $last_name = trim($_POST[‘last_name’]);

    }
if (empty($_POST[‘age’])) {

    $null_fields[] = ‘Age’;

    } else {

    $age = $_POST[‘age’];

    }
if (empty($_POST[‘gender’])) {

    $null_fields[] = ‘Gender’;

    } else {

    $gender = $_POST[‘gender’];
```

```
    }
if (empty($_POST[‘relationship’])) {

    $null_fields[] = ‘Reliationship’;

    } else {

    $relationship = $_POST[‘relationship’];

    }

if (empty($null_fields)) {

    $null_variable = NULL;
    $sql = “INSERT INTO FamilyMembers VALUES (‘$null_variable’, ‘$first_name’, ‘$last_name’, ‘$age’, ‘$gender’,
                                                    ‘$relationship’)”;

                                        if (!mysqli_query($conn, $sql)) {

                                                die (‘Error: ‘ .
mysqli_error($conn));

                                        }

echo “Family member has been entered!”;

mysqli_close($conn);
    }
                                    else {

                                        echo “You need to enter the follwing
```

```
missing data: <br />";

                                                    foreach ($null_fields as $null_field) {

                                                            echo $null_field . "<br />";

                                                    }
                                            }
    }

    ?>

<div>
<form action="member_added.php" method="post">
<div id="header">
<h3>Add Family Member</h3>
</div>

<div id="btn_add">
<a href="index.php">View Members</a>
</div>

<label for="fname">First Name</label>
<input type="text" id="first_name" name="first_name" maxlength="80">

<label for="lname">Last Name</label>
<input type="text" id="last_name" name="last_name" maxlength="80">

<label for="age">Age</label>
<input type="number" id="age" name="age" min="1" max="100">
```

```
<label for="gender">Gender</label>
<select id="gender" name="gender">
<option value="Male">Male</option>
<option value="Female">Female</option>
</select>

<label for="relationship">Relationship</label>
<input type="text" id="relationship" name="relationship" maxlength="80">

<input type="submit" name="submit" value="ADD">
</form>
</div>

</div>

</div>

</body>

</html>
```

Let us break this down and look at the most interesting bits.

We have to check if the information was submitted by the form before we work with it. We do this with an if statement and use the **isset()** built in PHP function that checks if a variable is null. We pass in the associative array **$_POST**as a parameter. It contains all values inputted into the form found in form.php. The array is passed to the script via the post method of the form:

```
if(isset($_POST['submit'])) {

}
```

Inside the conditional statement, we then declare an array to check if we happen to have missed any if the form inputs:

```
$null_fields = array();
```

Next we start checking if any the values in the array are empty before we send them to the database. If so, we add them to the **$null_fields** array if not, we create a variable and store them there. We will just look at one of the **if…else** statements because the rest a similar:

```
if (empty($_POST['first_name'])) {

    $null_fields[] = 'First Name';

} else {

    $first_name = trim($_POST['first_name']);

}
```

We do this for the rest of values in the **$_POST** array.

Warning: This may not be the most secure way of verifying data before it goes to the database. For that we need to use regular expressions and that is an advanced topic. We are just trying to keep things simple.

Next we have to check if the **$null_fields** array turned out empty. This means all the data was submitted:

if (empty($null_fields)) {

}

If we are inside that condition, it means that all the data was submitted and we can query the database. We use the **INSERT TO** statement to insert the data into the **FamilyMembers** table. We follow the rules used to insert data into a table by making sure the variables are in the same order as they are defined in the tables. We declared an extra variable before the creating the query string to pass in as a null value for the primary. We also make sure the number of parameters match the number of fields in the table:

$null_variable = NULL;

$sql = “INSERT INTO FamilyMembers VALUES (‘$null_variable’, ‘$first_name’, ‘$last_name’, ‘$age’, ‘$gender’, ‘$relationship’)”;

Next, we use query database using the **mysqli_query()**, passing in the connection and query string as parameters, while simultaneously checking for any errors return by the function. It returns **FALSE** on failure. If errors are found, we display them so they can be fixed. If not errors are found, we echo a success message:

if (!mysqli_query($conn, $sql)) {

die (‘Error: ‘ . mysqli_error($conn));

}else {

echo “Family member has been entered!”;

}

So now we are done with what happens when the player inputs all the data, but if he misses any, we should display to him the fields he missed by looping through the **$null_fields** array using a **foreach** loop and echoing them to the screen.

```
else {

    echo “You need to enter the follwing missing data: <br />”;

    foreach ($null_fields as $null_field) {

            echo $null_field . “<br />”;

    }
}
```

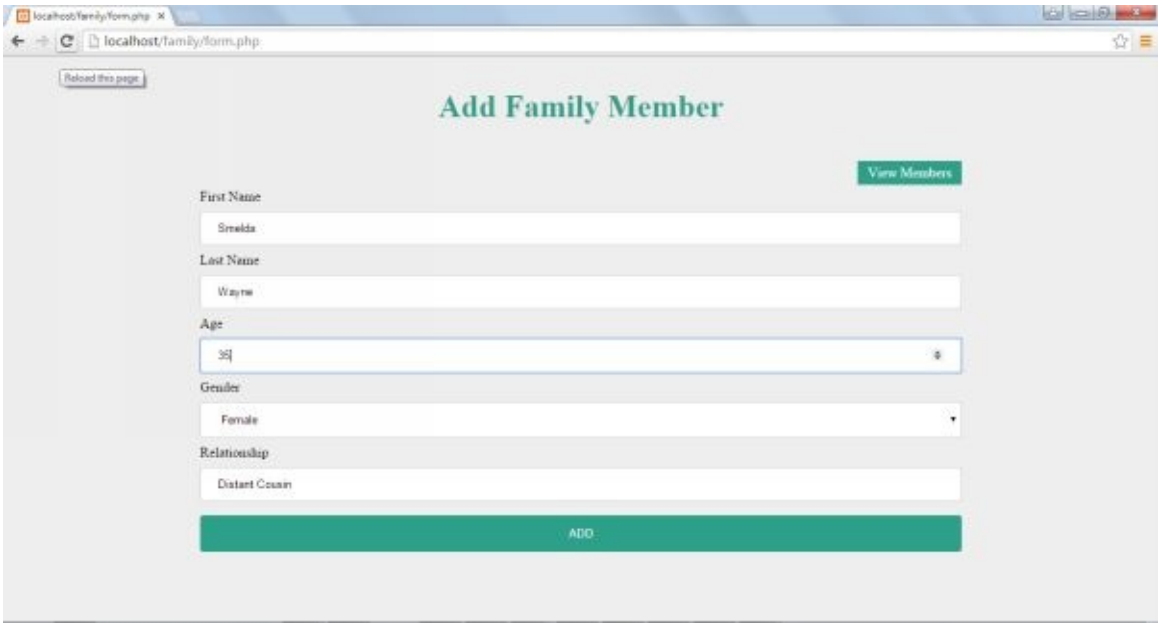
Then after that, we exit the condition that checks if the **$_POST** array is empty and close the connection to the database:

```
mysqli_close($conn);
```

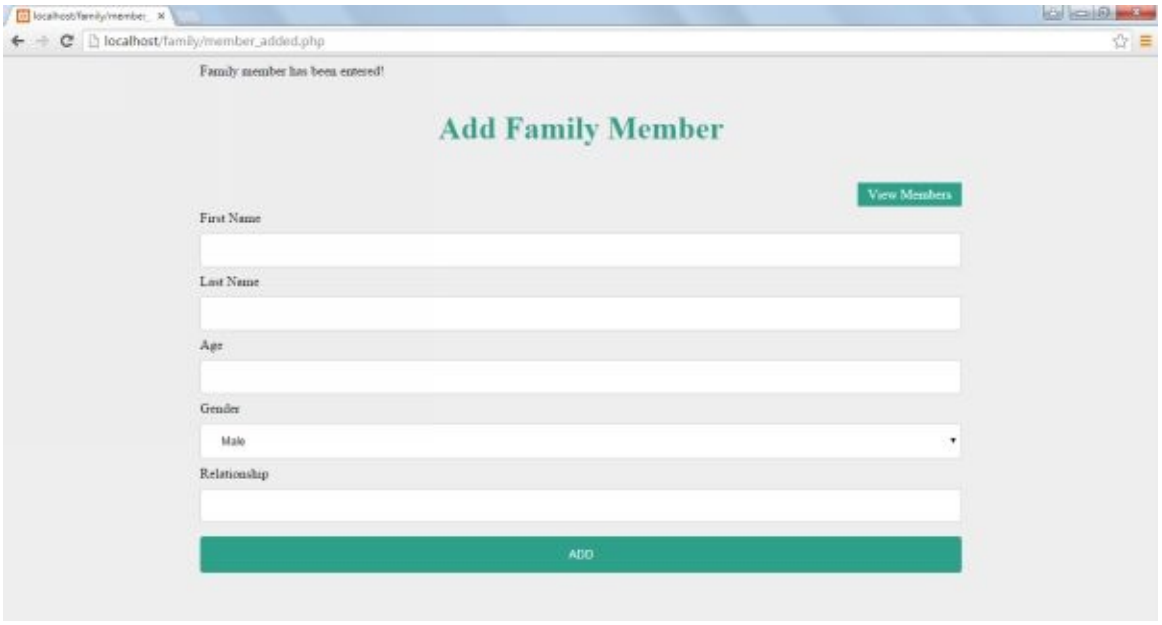
Then right after the closing PHP tag we display the form again that allows for the PHP script to call itself so that we can continue inputting data or re-enter data in case we forgot some fields.

And we are done! Now we should be able to add family members and view them. We should also be able to navigate between pages using the buttons on the top right corner of the forms or table that allow you switch between the pages.

Start up the index page and go to the form page to add some data



Click the ADD button to add the member to database. You should see the success message:



Click the View Members button to verify. The below window should display with the new member added:

Summary

So there you have it! Before this chapter we looked at PHP and MySQL separately and in this chapter was aimed at consolidating the two, and showing how to use them to produce dynamic content. It should not be that hard to modify the website to do whatever you need it to do as you are now capable.

In the chapter we created a configuration file that stored the connection to the database and used it all throughout the website to establish a connection to the database and query it to store and retrieve data. We created a page that enables you to view information about the family members you have added to database. We also created a page with a form for inputting the data and a page that has the PHP needed to process the data when the form is submitted.

Conclusion

“Education is the kindling of a flame, not the filling of a vessel.” — Socrates

This is the end of this book, but not your learning. You still have to go out there and learn what other things you can do with PHP and MySQL and get a more in depth understanding of the concept. At this point though, I sincerely hope you have been equipped with enough knowledge to do what is you need to do with a basic understanding of PHP and MYSQL.

In the first hour, we covered the basics of PHP like comments, variables, constants and operators. But not before we learned how to install the server package XAMPP to that has the Apache service for the server. We run a PHP script and displayed neat messages to the screen. We also chose a cool text editor to write beautiful PHP code.

In second hour, we got a little deeper into the basics of PHP and looked at how we can write conditional statements that allow us to execute blocks of code depending on a specified condition. Next we looked at how to loop thorough code until a certain condition is met using while and for loops. We then looked at how we encapsulated code blocks and re-use them through the program using functions. We passed information to the in the form of parameters and they return some of it back after performing operations on that information. Then we finally we saw how to store collections of data into arrays and how to loop through them.

In the third, hour we learned MySQL. We learned how to bring up the command prompt to enter MySQL commands to perform various operations such as creating and deleting databases; creating, altering and deleting tables; retrieving, inserting, updating and deleting the data in the tables.

In the fourth and final chapter we looked at how we can use PHP and MySQL together to create dynamic content for a simple website we were making a list of our favorite family members and their details.

All this is knowledge for the absolute beginner who needs something to quickly guide them with clear examples and explanations. I wish you well in your futures endeavors and I believe you are well on your way to achieving your dreams. Glad I could help.