HTML Form Elements - The Form Tags

- HTML forms are a way of gathering data from visitors to your page. Forms typically have text boxes for data input, radio buttons, drop down boxes, and buttons so that users can submit the form. A reset button is also handy, just in case mistakes are made filling out the form.
- For this section, create a new web page in your text editor. Save the file as **registrationForms.html.** We'll add the different form elements as we go along.

The Form Tag

- You don't have to tell your browser about any form elements on your web page. If you just want a simple form element like a text box, you can insert the text box tag by itself. But if you want to do something with the information on your form, like send it somewhere or to someone, you have to "tell" your browser about the form elements on your page. You do this with the Form tag:

```
<FORM>
</FORM>
```

- Like most HTML tags, the FORM tag comes as a pair, the forward slash preceding the second FORM tag to indicate that the form tag ends. Any form elements you need then go between these two FORM tags.

The NAME Attribute

- A NAME attribute is a useful addition to the FORM tag. When the form has a name, you can then refer to it in scripts.

  ```
  <FORM NAME = “regForm">
  </FORM>
  ```

- To give your form a name, type a space after FORM then type the word NAME, followed by an equals sign. Finally, add a name for your form. You can call it anything you like. Here, we've called the form **regForm.**

METHOD and ACTION Attribute

- If you want your form to go somewhere or to someone, two other attributes are needed in the FORM tag: METHOD and ACTION. Like this:

  <FORM NAME = “regForm" METHOD = "post" Action ="mailto:basiratabdusalam@gmail.com">

  </FORM>

The METHOD Attribute

- METHOD is way to send the data. There are two options:

1. Post
2. Get.

- Post sends the data in single lines of text;
- Get squashes all the data in a single line and adds it to the URL in the Action part. If the URL was an internet address, you'd see all the data you're sending in the address bar of your browser.

For instance, when u see this kind of thing in the URL:

registrationForm.html?text1=Barakat&text2=Ajani

Analysis

- The first name BArakat was typed into the text box called "text1" and the surname Ajani went into the text box called "text2".
- Form elements are separated by an ampersand (&). That is a direct result of using the Get method to send data.
- The **Post method doesn't add all that code to the address bar of your browser.**
- You should use Post to send your data, rather than Get.(**WHY?**)

The ACTION Attribute

- ACTION is used to specify the address where you want to send the data. For instance in the above, we're using an Email link to send the data straight to an email address:

  ACTION = basiratabdusalam@gmail.com

- But the form can be processed by a scripting language like CGI, ASP .NET, PHP, etc. In which case you'd specify the address of the script in question:

  ACTION = “regForm.php"

- To ensure that data in your forms is not sent anywhere, you can just add a pair of double quotes to the ACTION attribute:

  ACTION = ""

Form Elements

There are quite a few different form elements you can add to a form:

Text Boxes, Text Areas, Option Buttons ,Check Boxes, Drop down List/Fixed Lists, Password Boxes, Command Buttons, Submit Buttons, Reset Buttons, Labels, Image Command Buttons Hidden Form Values.

- HTML5, however, adds even more:

Placeholder, Email, Url, Number, Range , Date/Time ,Search, Color.

Note

- However, note that some of these new form elements are only supported in certain browsers. For example, Color refers to a popup colour picker. Only the Opera web browser supports this, at the time of writing.

The INPUT Tag and Text Boxes

- B4 we talk about the form elements, lets first talk about the input tag. For most form elements, the word INPUT is used to set up the element. Next, you type a space followed by the word TYPE. This tells the browser what type of form elements to draw on your page. If you want a text box, the TYPE to use is "Text":

```
<INPUT TYPE = "Text">
```

Note that there is no end tag for INPUT.

- Next, you add the attributes you want. There are quite a lot of attributes for text boxes. The three most common ones are Size, Value, and Name. Size refers to how long the text box is in pixels rather than the number of characters it can contain. It's a good idea to limit how many text characters can fit into your text box, and for this the Maxlength attribute is used:

```
<INPUT TYPE = "Text" Size = 20 Value = "" Name = "text1"
    Maxlength="15">
```

- The Value attribute means the text that will be typed in your text box. You can set some default text, if you like:
- <INPUT TYPE = "Text" Size = 20 Value = "Default Text Here" Name = "text1">
- Whatever is between the two quote marks of VALUE is what you get back when the form is sent somewhere.
- A **Name** attribute should be added so that you can refer to it in scripts. It distinguishes it from any other text box you may have on your form.

Some other text box attributes are:

Accesskey, Height ,TabIndex ,Width

- Height and Width are self-explanatory.
- Accesskey refers to a keyboard shortcut you can add.
- TabIndex is a number value and refers to where the cursor will end up when you press the Tab key on your keyboard. As an example, take a look at the following three text boxes:
- **<INPUT TYPE = "Text" TabIndex="2">**
  <INPUT TYPE = "Text" TabIndex="3">
  <INPUT TYPE = "Text" TabIndex="1">
- The third of the three text boxes above will be activated first when the tab key is pressed. Press the tab key again and the cursor will appear in the top text box, followed by the middle one. You can really annoy your visitors if you get the tab index wrong!

The Submit Button

- If you want your form to be sent somewhere, a Submit button is needed (though you can write code for your own submit button). When the Submit button is clicked, the browser will check the ACTION attribute of the FORM tag to see where you want the data sent. It then checks the METHOD attribute to see what method you want to use, Post or Get. The browser will then try to send the form's data for you.
- The code for a Submit button is this:

<INPUT TYPE = "Submit" Value = "Submit">

- Here, the TYPE is set to "Submit".
- The VALUE attribute is the text that will appear on the button itself. The width of the button is determined by the width of the text for the VALUE attribute.
- If you wanted a really wide button, you can use this old trick by adding spaces to the left and right of the text:

Value = " Submit ">

- The browser will see the spaces as text and adapt the width accordingly.

The Reset Button

The Reset button returns the form to the state it was originally in when it was loaded. Any default values you added will be retained. The code for a rest button is this:

<INPUT TYPE = "Reset" Value = "Clear">

- Note the TYPE is now "Reset". The value attribute works in the same way that the Submit button does - it's the text that will appear on the button.
-

To do

- To test out the text box element and the buttons, add the following between the BODY tags of your HTML code:

```
<FORM>
   <INPUT TYPE="Text">
   <P>
   <INPUT TYPE="Submit" Value="Submit">
</FORM>
```

- The form is just a text box and a button. Notice that the two are separated by a P tag, otherwise they'd be side-by-side.
- Save your work and view the results in your browser. You should see this:

- The text box will be longer and your default text will display.

- Type something into your text box and click the button. What you'll find is that the text will disappear. If you had added Method and Action attributes to the FORM tag then the text box data would be sent to a location of your choice (email address, script, etc).

METHOD attribute:

<FORM NAME ="form1" METHOD =" " ACTION = "">

- The **Method** attribute is used to tell the browser how the form information should be sent.
- The two most popular methods you can use are GET and POST. But our METHOD is blank. So change it to this:

  <FORM NAME ="form1" **METHOD ="GET"** ACTION = "">

  To see what effect using GET has, save your work again and then click the Submit button on your form. You should see this:

- The thing to notice here is the address bar. After **basicForm.php**, we have the following:

  ?Submit1=Login
- This is a consequence of using the GET method. The data from the form ends up in the address bar.
- You'll see a question mark, followed by form data.
- In the image above, **Submit1** was the NAME of the button, and **Login** was the VALUE of the button (the text on the button). This is what is being returned by the GET method. You use the GET method when the data you want returned is not crucial information that needs protectiON.

The Post Attribute

- The alternative to GET is to use POST. Change the first line of your FORM to this:

<FORM NAME ="form1" **METHOD ="POST"** ACTION = "">

- Close your browser down, and open it back up. Load your basicForm.php page again, and then click the button. Your address bar will then look like this:

- The ?Submit1=Login part from the previous section is now gone! That is because we used POST as the method.
- Using POST means that the form data won't get appended to the address in the address bar for all to see.
- Using POST and GET depends on the project: if the data is not sensitive then use GET, otherwise use POST.

Action attribute

- Another important attribute of the Form tag is Action. Without Action, your forms won't go anywhere!
- The Action attribute is crucial. It means, "Where do you want the form sent?". If you miss it out, your form won't get sent anywhere.
- You can send the form data to another PHP script, the same PHP script, an email address, a CGI script, or any other form of script.
- In PHP, a popular technique is to send the script to the same page that the form is on – send it to itself, in other words. We'll use that technique first, but you'll see both techniques in action.

- Change the form you have been creating in the basicForm.php. Locate the following, and amend the ACTION line to this:
- <Form Name ="form1" Method ="POST" **ACTION = "basicForm.php"**>
- So we're going to be sending the form data to exactly the same page as the one we have loaded – to itself. We'll put some PHP on the page to handle the form data. But for now, save your work again and then click your submit button. You won't see anything different, but you shouldn't see any error message either!

The Submit Button of HTML Forms

- The HTML Submit button is used to submit form data to the script mentioned in the ACTION attribute.

<Form Name ="form1" Method ="POST" **ACTION = "basicForm.php"**>

- So the page mentioned in the ACTION attribute is basicForm.php.
- To Submit this script, you just need a HTML Submit button:

<INPUT TYPE = "Submit" Name = "Submit1" VALUE = "Login">

- You don't need to do anything special with a Submit button – all the submitting is done behind your back. As long as SUBMIT has an ACTION set, then your data will get sent somewhere. But the NAME attribute of the Submit buttons comes in very handy. You can use this Name to test if the form was really submitted, or if the user just clicked the refresh button. This is important when the PHP script is on the same page as the HTML form. Our Submit button is called "Submit1", but you can call it almost anything you like.

APACHE, PHP,and MySQL

In creating our website, imagine you wanted to create a dynamic restaurant for “*Gbadun Emi e*” . Customers come to your place, and each one wants something different and specific. They don’t worry so much about how the food is prepared, as long as it looks and tastes delicious. Unlike a buffet-type spread, where everything is laid out and your

Customers simply choose from what’s available, a nice restaurant encourages customer/waiter interaction

and complete customization for any specific dietary needs. Similarly, a Web site shouldn’t be a static page with little interaction from visitors; it should be a dynamic site where the visitor can choose what he or she wants to see.

In this scenario, you can characterize the three components of the AMP module as follows:

- **Apache:** This is your highly trained master, *the chef*. Whatever people ask for, she prepares it without complaint. She is quick, flexible, and able to prepare a multitude of different types of foods. Apache acts in much the same way as your HTTP server, parsing files and passing on the results.
- **PHP:** This is the waiter. He gets requests from the patron and carries them back to the kitchen with specific instructions about how the meal should be prepared.
- **MySQL:** This is your stockroom of ingredients (or in this case, information).

Passing Variables between Pages

Suppose your site allows viewers to enter their name on the front page. You’d like to be able to greet the user by name on each page in your site, but to do so, you need some way to pass the value of the name variable from page to page.

There are basically four ways to accomplish this task: pass the variables in

- The URL, through a session, via a cookie, or with an HTML form. The method you choose is based on the situation and what best fits your needs at the time.

Passing Variables through a URL

The first method of passing variables between pages is through the page's URL. You've undoubtedly

seen URLs such as this:

http://www.fuo.edu.ng/news/articles/webdevelopment.php?id=123

This is an example of passing variable values through the URL. It requests that the article with the ID number of "123" be chosen for the webdevelopment.php program. The **text after the URL is called** the *query string*.

You can also combine variables in a URL by using an ampersand (&), as in this example:

http://www.fuo.edu.ng/news/articles/webdevelopment.php?id=123&lang=en

This asks to retrieve the file with an ID of "1234" and the language presumably equal to "en," for English. There are a few disadvantages to passing variables through a URL:

❑ Everyone can see the values of the variables, so passing sensitive information isn't really very secure using this method.

❑ The user can change the variable value in the URL, leaving your site potentially open to showing something you'd rather not show.

❑ A user might also pull up inaccurate or old information using a saved URL with older variables embedded in it.

APACHE, PHP,and MySQL

- PHP, Apache, and MySQL are all part of the *open source* group of software programs.
- By allowing the open exchange of information, programmers from all over the world contribute to make a truly powerful and efficient piece of software available to everyone. Through these, publicly available source code, bugs get fixed, improvements are made, and a good software program becomes a great one over time.

Passing Variables with Sessions

Passing a value through a URL is fine if the information is not of a **particularly sensitive nature** or **if it is relatively static** and there **is no danger** of a user pulling up old information from a previously saved page.

If you are transmitting information such as usernames or passwords, however, or personal information such as addresses and phone numbers, better methods exist for passing the information while keeping it private.

A *session* is **basically a temporary set of variables that exists only until the browser has shut down.**

Examples of session information include a session ID and whether or not an authorized person has "logged in" to the site. This information is stored temporarily for your PHP programs to refer back to whenever needed.

Every session is assigned a unique session ID, which keeps all the current information together. Your

session ID can either be passed through the URL or through the use of cookies. **Although it is preferable**

for security reasons to pass the session ID through a cookie so that it is hidden from the human eye, if

cookies are not enabled, the backup method is through the URL.

- To begin a session, use the function session_start().

Suppose you want to pass your visitor's username, and whether or not he or she has authentically logged into the site between the first page and the second page

The Syntax is : $_SESSION['varname']

findbook.php

```
<?php
session_start();
$_SESSION[‘username’] = “Baseerat”;
$_SESSION[‘authuser’] = 1;
?>
<html>
<head>
    <TITLE>Find my Favourite Book!</TITLE>
</head>
<body>
      <?php
          $favouritebook = “  ALQURAN”;
           echo “<a href=’favouritebook.php?favouritebk=$favouritebook ’>”;
           echo “Click here to see information about my favorite book!”;
           echo “</a>”;
       ?>
</body>
</html>
```

favouritebook.php

```
 <?php
      session_start();
       //check to see if user has logged in with a valid password
       if ($_SESSION['authuser'] != 1) {
             echo "Sorry, but you don't have permission to view this page!!!";
             exit();
              }
  ?>
<html>
<head>
<title>My favouritebook- <?php    echo $favouritebook;    ?></title>
</head>
<body>
   <?php
          echo "Welcome to our site, ";
          echo $_SESSION['username'];
          echo "! <br>";
          echo "My favourite book is ";
          echo $favouritebook;
          echo "<br>";
          $bookrate = 500;
          echo "The rate for this book is: ";
          echo $bookrate;
  ?>
</body>
```

Important Notes

1. All the session information is at the top of the page, before any HTML code. This is very important!
2. You must use the function session_start() at the beginning of every page that references the session variables.
3. We have used an “if statement”, which you can delve into later . It’s a good idea to take a glance at this syntax, just to familiarize yourself with it.

Passing Information with Forms

Up until now, you've passed information among pages successfully, but you've been the one to supply all the information, not the visitor. Although it would be a great world if you really knew that much about your Web site visitors, it might get a little labor-intensive on your part. What do you say to letting them supply you with information for a change?

Forms are the great Venus Fly Traps, just lying in wait to gobble up useful information from Web site visitors. Forms allow your Web site to be truly interactive; they take data from the user and send it off somewhere where it gets massaged, manipulated, perhaps stored, and then some result is sent back to the user. We will briefly touch

on them here so you get a basic understanding of how they work.

How to use forms to get information from visitors:

findbook.php

```
<?php
 session_start();
    $_SESSION[‘username’] = $_POST[‘user’];
    $_SESSION[‘userpass’] = $_POST[‘pass’];
    $_SESSION[‘authuser’] = 0;
//Check username and password information
    if (($_SESSION[‘username’] == ‘Baseerat’) and  ($_SESSION[‘userpass’] == ‘123’))
      {
        $_SESSION[‘authuser’] = 1;
      }
     else {
           echo “Sorry, but you don’t have permission to view this  page!!!”;
      exit();
      }
?>
<html>
<head>
  <TITLE>Find my Favourite Book!</TITLE>
</head>
<body>
    <?php
       $favouritebook = “  ALQURAN”;
        echo “<a href=’favouritebook.php?favouritebk=$favouritebook ’>”;
        echo “Click here to see information about my favorite book!”;
        echo “</a>”;
     ?>
</body>
</html>
```

favouritebook.php

```
<?php
 session_start();
 //check to see if user has logged in with a valid password
   if ($_SESSION['authuser'] !=1 ) {
        echo "Sorry, but you don't have permission to view this page!!!";
    exit();
     }
  ?>
<html>
<head>
<title>My favouritebook- <?php    echo  $favouritebook;    ?></title>
</head>
<body>
   <?php
          echo "Welcome to our site, ";
          echo $_SESSION['username'];
          echo "! <br>";
          echo "My favourite book is ";
          echo $favouritebook;
          echo "<br>";
          $bookrate = 500;
          echo "The rate for this book is: ";
          echo $bookrate;
  ?> </body>
</html>
```

Start a new file: login.php

```
<?php
session_unset();
?>
<html>
<head>
<title>Please Log In</title>
</head>
<body>
    <form method=”post” action=”findbook.php”>
    <p>Enter your username:
    <input type=”text” name=”user”>
   </p>
     <p>Enter your password:
     <input type=”password” name=”pass”>
     </p>
     <p>
      <input type=”submit” name=”Submit” value=”Submit”>
     </p>
</form>
</body>
</html>
```

- Save this file as login.php.
- Load the login.php file into your browser and log in with the username Baseerat and the password 123.

How It Works

- In login.php, you first release any variables from sessions that may be lingering around with the command session_unset().
- Then you ask for two variables from the user: username and password (variable names user and pass, respectively). These are submitted to findbook.php (the "action" in the form) via the POST method (the "method" in the form). This is why you have to refer to them using the $_POST syntax at the beginning of findbook.php.

The file findbook.php actually accomplishes several things:

- It starts the session and, by default, registers the variables. Values are set based on the information sent from the form in login.php.
- It checks to see if the username and password are acceptable. In real life, you would match this information to a database for authentication and verification.
- It sets the authuser to 1 if the acceptable username/password combination has been supplied, which grants the user permission to then proceed to other pages in the site, such as favouritebook.php.
- If the username/password combination is not acceptable, a tactful error message is displayed to the user.
- Because the information is passed on to favouritebook.php as before, the only thing favouritebook.php has to check for is that the user is authorized through the authuser variable.

Using Includes for Efficient Code

The makers of PHP have blessed developers with a little time-saving device called “includes” that save you from reentering frequently used text over and over.

Suppose that you want to type the same message on every page of your site. Perhaps it is your company’s name and address, or maybe today’s date. If you are coding each page of your site from scratch, this is not very efficient for a couple of reasons:

You are typing the same information over and over again, which is never good. In the case of an update or a change, you have to make the change in every single page of your site. Again, this is redundant and time consuming, and it elevates the potential for human error. A solution to this problem is to use an include.

Includes are PHP files tucked into other PHP files. You take commonly used information and put it in a separate file. For example, if you have a set of defined variables that need to be referenced in every page on your site, you could define them once, in a single PHP script. Then, on each of your pages where you want the variables to appear, you use an include statement that specifies the file that defines the variables. When your script is parsed, the parser inserts the code from the include file into your page, just as if you’d typed it there yourself. The final output is then sent to the browser.

Date function

```
<div align=”center”><font size=”4”>Welcome to my book review site!</font>
<br>
 <?php
    echo “Today is “;
    echo date(“F d”);
    echo “, “;
    echo date(“Y”);
  ?>
  </div>
```

Processing the text the user entered into the text boxes

- Recall that the METHOD attribute tells you **how** form data is being sent, and the ACTION attribute tells you **where** it is being sent.
- To get at the text that a user entered into a text box, the text box needs a NAME attribute. You then tell PHP the NAME of the textbox you want to work with. So far, our text box hasn't got a NAME yet, so change your HTML to this:

<INPUT TYPE = "Text" VALUE ="username" NAME = "username">

- The NAME of our textbox is **username**. It's this name that we will be using in a PHP script.
- To return data from a HTML form element, you use the following syntax:

$_POST['formElement_name'];

You can assign this to a variable:

$Your_Variable = $_POST['formElement_name'];

Add the following PHP script to the HEAD section of ur HTML code you have so far. U will have this:

```
<html>
   <head>
   <title>A BASIC HTML FORM</title>
    <?php
        $username = $_POST['username'];
        print ($username);
    ?>
</head>
```

- Save your work again, and click the submit button to run your script. (Don't worry if you see an error message about "Undefined index". Click the button anyway.) You should see this appear above your text box:

How does it work?

- The **$_POST[ ]** is an inbuilt function you can use to get POST data from a form. If you had METHOD = "GET" on your form, then you'd used this instead:

  $username = $_GET['username'];
- So you begin with a dollar sign (**$**) and an underscore character (**_**). Next comes the METHOD you want to use, POST or GET. You need to type a pair of square brackets next. In between the square brackets, you type the NAME of your HTML form element – **username**, in our case.

  $_POST['username'];
- Note: you put semi-colon to complete the line.

- Whatever the VALUE was for your HTML element is what gets returned. You can then assign this to a variable:

  $**username = $_POST['username'];**

- So PHP will look for a HTML form element with the NAME **username**. It then looks at the VALUE attribute for this form element. It returns this value for you to use and manipulate.

Using a bit of Conditional Logic to test what is inside of the variable

- At the moment, all we're doing is returning what the user entered and printing it to the page. But we can use a bit of Conditional Logic to test what is inside of the variable. As an example, change your PHP to this:

```
        $username = $_POST['username'];
            if ($username == “Baseerah") {
                  print ("Welcome back” + $username  + “!");
            }
      else {
                  print ("You're not authorized to view this site");
            }
```

- We're now checking to see if the user entered the text “Baseerah". If so, the username is correct; if not, print another message.

Checking if the submit button was clicked

- In the previous section, you saw how to get text from a textbox when a Submit button on a form was clicked. However, when you first load the page the text still displays.
- The reason why the text displays when the page is first loaded is because the script executes whether the button is clicked or not. This is the problem you face when a PHP script is on the same page as the HTML, and is being submitted to itself in the ACTION attribute.

- To get round this, you can do a simple check using another IF Statement. What you do is to check if the Submit button was clicked. If it was, then run your code. To check if a submit button was clicked, use this:

  if (**isset(** $_POST['Submit1']) **)** { }
- Don’t worry as it looks a bit confusing! But it actually consists of three parts:

if ( ) { }
isset( )
$_POST['Submit1']

You know about the if statement. But in between the round brackets, we have **isset( )**. This is an inbuilt function that checks if a variable has been set or not. In between the round brackets, you type what you want isset( ) to check. For us, this is **$_POST['Submit']**. If the user just refreshed the page, then no value will be set for the Submit button. If the user did click the Submit button, then PHP will automatically return a value. Change you script from the previous work to the following and try it out:

```
 if (isset($_POST['Submit1'])) {
     $username = $_POST['username'];
       if ($username == “Baseerah") {
         print ("Welcome back” + $username  + “!");
         }
     else {
             print ("You're not a member of this
site");
         }
       }
```

Using PHP with MySQL

- Understanding MySQL database
- Viewing data contained in the MySQL database
- Connecting to the database from your Web site
- Pulling specific information out of the database, right from your Web site

Overview of MySQL Structure and Syntax

- MySQL is a relational database system, which basically means that it can store bits of information in separate areas and link those areas together. You can store virtually anything in a database: the contents of an address book, a product catalog, or even a wish list of things you want for your birthday.

MySQL Syntax and Commands

- Here, we are going to access MySQL through PHP. Common SQL commands include:
- CREATE: Creates new databases and tables
- ALTER: Modifies existing tables
- SELECT: Chooses the data you want
- DELETE: Erases the data from your table
- DESCRIBE: Lets you know the structure and specifics of the table
- INSERT INTO *tablename* VALUES: Puts values into the table
- UPDATE: Lets you modify data already in a table
- DROP: Deletes an entire table or database.

MySQL commands within PHP

Some of the more commonly used functions are:

1. mysql_connect ("*hostname*", "*user*", "*pass*"): Connects to the MySQL server.
2. mysql_select_db("*database name*") Equivalent to the MySQL command USE; makes the selected database the active one.
3. mysql_query("*query*"): Used to send any type of MySQL command to the server.
4. mysql_fetch_rows("*results variable from query*"): Used to return a row of the entire results of a database query.
5. mysql_error(): Shows the error message that has been returned directly from the MySQL server.

Connecting to the Database

Before you can connect to the database, you need four things:

- ❑ Database name
- ❑ Host Server name
- ❑ Username
- ❑ Password

Connect to the database using the following command (in PHP):

$connection = mysql_connect("*servername*", "*username*", "*password*");

You then need to select the appropriate database by using the following command (in PHP):

```
<?php
    $database = mysql_select_db("databasename", $connection)
    or die("couldn't find the database");
?>
```

You can also send any MySQL command to the server through PHP and the mysql_query command, as in the preceding example. You do this by sending the straight text through PHP either through a variable or through the mysql_query command directly, like this:

```
$query = “SELECT * from TABLE”;
$results = mysql_query($query);
```

You can also do it like this:

```
$results = mysql_query(“SELECT * from TABLE”);
```

Connecting to the MySQL Server

Before you can do anything with MySQL, you must first connect to the MySQL server using your specific connection variables. Connection variables consist of the following parameters:

1. **Host name:** It's the local host because you've installed everything locally. You will need to change this to whatever host is acting as your MySQL server.
1. **Username and password:** We're going to use a new username that we created for use with the examples throughout the rest of the book.

You issue this connection command with the PHP function called mysql_connect. As with all of your PHP/MySQL statements, you can either put the information into variables, or leave them as text in your MySQL query.

Here's how to do it with variables:

```
$host = "localhost";
$user = "root";
$pass = "";
$connect = mysql_connect($host, $user, $pass);
```

The following statement has the same effect:

```
$connect = mysql_connect("localhost", "root", "");
```

Creating a Database

Lets create the database and tables that you can use in the website uhave created for your favourite book
1. Open your browser and type the following code. This creates your database and the tables you need to hold the data.

```
<?php
   //connect to MySQL  with your own for hostname, user, and password
          $connect = mysql_connect(“localhost”, “root”, “”) or
         die (“Pls check your server connection.”);
  //create the main database if it doesn’t already exist
         $create = mysql_query(“CREATE DATABASE IF NOT EXISTS booksite”)
          or die(mysql_error());
  //make sure our recently created database is the active one
          mysql_select_db(“booksite”);
   //create “book” table
         $book = “CREATE TABLE book (
         book_id int(11) NOT NULL auto_increment,
         book_name  varchar(255) NOT NULL,
         book_type tinyint(2) NOT NULL default 0,
         book_year int(4) NOT NULL default 0,
         book_publisher varchar(255)  NOT NULL default 0,
         book_reviewer varchar(255)  NOT NULL default 0,

         PRIMARY KEY (book_id),
$results = mysql_query($book)
or die (mysql_error());
//create “booktype” table
    $booktype = “CREATE TABLE booktype (
    bookype_id  int(11) NOT NULL auto_increment,
    booktype_label  varchar(100) NOT NULL,
    PRIMARY KEY (booktype_id)
    )”;
$results = mysql_query($booktype)
or die(mysql_error());
echo “Book Database successfully created!”;
?>
```

 Save this file as createbook.php.

Create a new file, and name it bookdata.php.(The file that will populate the database)

```
<?php
//connect to MySQL
      $connect = mysql_connect(“localhost”, “bp5am”, “bp5ampass”)
      or die (“Hey loser, check your server connection.”);
//make sure we’re using the right database
      mysql_select_db(“booksite”);
//insert data into “book” table
      $insert = “INSERT INTO book (book_id, book_name, book_type, “ .
      “book_year, book_publisher , book_reviewer) “ .
      “VALUES (1, ‘The Mighty’, ‘Novel’, 2003, ‘Adam Jembola’, ‘Alao Bas’), “ .
      “(2, ‘Return of Pharaoh’, ‘Novel’, 2012, ‘Akande Qudus’, ‘Ojo Smith), “ .
      “(3, ‘Loving Yourself’, ‘Novel’, 2014, ‘Yusuf Taofeek’, ‘Aolat Helen)”;
      $results = mysql_query($insert)
      or die(mysql_error());
      //insert data into “booktype” table
      $type = “INSERT INTO booktype (booktype_id, booktype_label) “ .
      “VALUES (1,’Sci Fi’), “ .
      “(2, ‘Islamic’), “ .
      “(3, ‘Textbook’), “ .
       $results = mysql_query($type) or die(mysql_error());
```

Querying the Database and Using the SELECT Query

Now that you have some data in the database, you probably want to retrieve it. You use the SELECT statement to choose data that fits your criteria.

Typical syntax for this command is as follows:

```
        SELECT [fieldnames]
        AS [alias]
        FROM [tablename]
        WHERE [criteria]
        ORDER BY [fieldname to sort on] .
```

Open your text editor and type this code and save this file as select.php:

```
<?php
//connect to MySQL
        $connect = mysql_connect(“localhost”, “bp5am”, “bp5ampass”)
        or die(“Hey loser, check your server connection.”);
//make sure we’re using the right database
        mysql_select_db(“booksite”);
        $query = “SELECT book_name, book_type “ .
        “FROM book “ .
        “WHERE book_year>2012 “ .
        “ORDER BY book_type”;
        $results = mysql_query($query)
        or die(mysql_error());
?>
```