**Lecture No.26**

**Passing variables between pages**

In a web page we can have variables. These variables are available as long as we remain on that page. When we move to some other page, we cannot find those variables. However, PHP provides us some super global variables that can keep variables and we can retrieve these variables on other pages. In this lecture we will learn how to pass variables between pages.

1. **Passing form’s data:**

Forms provide a mean of submitting information from the client machine to the server. We can create HTML forms using <form> tag. Method and action are the most common attributes of the <form>. Action is used to give the URL of the application that is to receive and process the forms data. Method sets the HTTP method that the browser uses to send the form's data to the server for processing. Get and post are the common methods. In get method all form data is encoded into the URL, appended the action URL as query string parameters while in post method form’s data appears within the message body of the HTTP request.

* 1. **Super global variables:**

PHP automatically makes few variables available in your program. These are array variables and can be accessed by name. These variables are called super-global variables because they can be accessed without regard to scope. The most commonly used super global variables are $\_GET and $\_POST. Below we discuss these super global variables in detail.

**$\_GET super global variable:** it contains all the query string variables that were attached to the URL. As we discussed earlier, that when we use get method in a form, its input is transferred along with the URL as query string parameters therefore, $\_GET variable also keeps the input of the from when it is sent by get method. On the action page we can retrieve user’s input from $\_GET variable.

We show this mechanism by the following example. Consider the following code for a form

<body>

<form method=“get” action=“action.php”>

<input type=“text” name=“name”>

<input type=“text” name=“email”>

<input type=“submit”>

</form>

</body>

This code creates a form with two input elements named ‘name’ and ‘email’. When a user submits this form after providing its input the user’s input is saved in the $\_GET super global variable. These values are stored in the $\_GET super global array by name. The names of the cells are the same as of the input fields in the form.

**email**

**name**

**$\_GET**

**User’s email address**

**User’s name**

We can retrieve these values on the action page from the $\_GET array. For example, if we want to retrieve the user’s name we can do this by $\_GET[‘name’].

**$\_POST super global variable:** $\_POST is also an array type variable which is created automatically by PHP. It contains all the submitted form variables and their data. When we use post method in the form, form’s input data is stored in the $\_POST super global array which can be retrieved on the action page. If we consider the form code of the previous example and set the value of the method to post in the form tag, the user’s input is stored in the $\_POST array on the submission of the form. It is shown in the following diagram.

**email**

**name**

**$\_POST**

**User’s email address**

**User’s name**

On the action page we can retrieve user’s input from $\_POST variable. For example, if we want to retrieve the user’s name we can do this by $\_POST[‘name’].

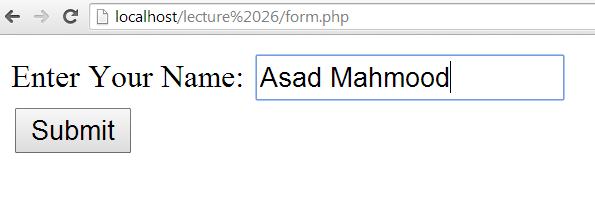
* 1. **Processing form’s elements:**

In the section we will discuss how we can retrieve the value of different form elements on the action page.

**Textbox:** when the user submits a form, values of all text boxes is stored in $\_GET or $\_POST super global arrays depending upon the method we used in the form. These values can be retrieved by the name of the input field in the form. In the following example we creates a form which asks the user to enter its name and a submit button. The action page gets the user’s input and displays a welcome message.

|  |  |
| --- | --- |
| **form.php**  <html>  <head>  <title>Passing Form Data</title>  </head>  <body>  <form id="form1" name="form1" method="post" action="action.php">  Enter Your Name:  <input type="text" name="name" value="Your Name" /><br>  <input type="submit" />  </form>  </body>  </html> | **action.php**  <?php  $name = $\_POST['name'];  echo "Welcome Mr. $name";  ?> |

The output of the above program is show in following figures. In the first figure user enter its name and submits the form while the second diagram shows the action page with a welcome message.

**Hidden field:** hidden fields are not visible to the user but their value can be retrieved on the action page. We retrieve the value of a hidden field from $\_GET or $\_POST. Suppose a form sends its input data to the server using get method and this form has a hidden input field with name “hvalue”. We can retrieve the value of this hidden field on the action page by $\_GET[‘hvalue’].

**Checkbox:** we can retrieve the value of a checkbox on the action page by its name. If a user checks a checkbox and submits the form the value of the checkbox is sent to the server. If the value of the checkbox is not set in the input tag then ‘on’ is sent to the server

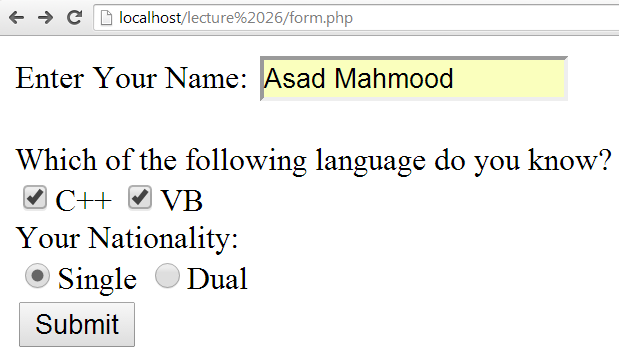
**Radio button:** Radio buttons shows some options to the user and the user can select only one of them. We can retrieve the value of the selected radio button on the action page by its name.

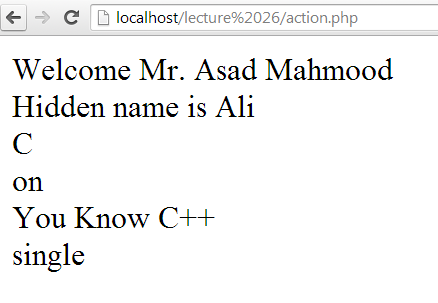
In the following example we example we have used the form of the previous example and added the following fields in it

* A hidden field named ‘hname’ and its value is set to Ali on the action page we retrieved this value and displayed on the screen
* Two check boxes with name ‘C’ and ‘VB’. The value of checkbox C is not set while the value of the checkbox VB is set to VB. On the action page the value of both check boxes is displayed (the value will be displayed if the user checks the checkbox before submitting the form). We have displayed a message if the user checks the C checkbox.
* A group of radio button asking about the nationality of the user. The value of the selected radio button is also displayed on the browser

|  |  |
| --- | --- |
| <html>  <head>  <title>Passing Form Data</title>  </head>  <body>  <form id="form1" name="form1" method="post" action="action.php">  Enter Your Name: <input type="text" name="name"  value="Your Name" /><br>  <input type="hidden" name="hname" value="Ali" /><br />  Which of the following language do you know? <br />  <input type="checkbox" name="C" value="C" />C++  <input type="checkbox" name="VB" />VB  <br />  Your Nationality: <br />  <input type="radio" name="nat" value="single" />Single  <input type="radio" name="nat" />Dual  <br />  <input type="submit" />  </form>  </body>  </html> | <?php  $name = $\_POST['name'];  echo "Welcome Mr. $name";  echo "<br>";  echo "Hidden name is ". $\_POST['hname'];  echo "<br>";  echo $\_POST['C'];  echo "<br>";  echo $\_POST['VB'];  echo "<br>";  if($\_POST['C']=='C')  echo "You Know C++";  else  echo "You do not know C++";  echo "<br>";  echo $\_POST['nat'];  echo "<br>";  ?> |

The output of the above program is show in the following figures





**Select list:** select list provided multiple options to a user. The value of the selected list item is passed to the server when the user submits the form and on the action page we can retrieve this value by name of the select list.

1. **Passing data using sessions**

A session is basically a temporary set of variables that exists only until the browser has shut down. These variables are accessible on other pages. While sending data on other pages using sessions, we store data in the $\_SESSION super global array and can retrieve the stored data on any other page. $\_SESSION super global array works as a common memory. We store data from different pages in this array. The stored data can be accessed on any other page.

Following are some basics functions used while sending data using sessions

* **session\_start()-** is used to start a session. This function is used on every page where we use sessions. This function is called at the top of the page even before the <HTML> tag.
* **$\_SESSION[‘variable\_name’]-** is used to store data in session variable
* **session\_destroy()-** is used to destroy a session
* **unset($\_SESSION[‘variable\_name’])-** is used to unset a specific variable

**In the following example we have created the following pages**

* on first page we start a session. In the body section we create a link to another page and a session variable $\_SESSION[‘name’] is declared which can be accessed on other pages.
* on second page we again start the session. In the body section we display the value of the session variable declared/stored in the previous page and a link to the third page
* On third page we start the session and destroy the session. It means the previously stored values are removed from the session array. In body section we display the value of the session variable (as values is removed from the array therefore we could not get that value)

|  |  |  |
| --- | --- | --- |
| <?php  session\_start();  ?>  <html>  <head>  <title>Using Sessions</title>  </head>  <body>  <?php  $\_SESSION['name']='Ali';  ?>  <h1>Welcome to the first page </h1>  <a href="session2.php">Go to the next page</a>  </body>  </html> | <?php  session\_start();  ?>  <html>  <head>  <title>Getting Session</title>  </head>  <body>  <h1> This is the second page</h1>  <?php  echo "The value received from sessoin variable is ". $\_SESSION['name'];  ?><br>  <a href="session3.php">Go to the third page </a>  </body>  </html> | <?php  session\_start();  session\_destroy();  ?>  <head>  <title>Destroying Session</title>  </head>  <body>  <?php  echo $\_SESSION['name'];  ?>  </body>  </html> |

**References:**

Chapter 2, “Beginning PHP6,Apache,Mysql web development” by Matt Doyle, Wrox publishers, 2009, ISBN: 0470413964

Chapter 13, “Beginning PHP and MySQL” by W. Jason Gilmore, Apress publisher, 4th edition; 2010, ISBN-13 (electronic): 978-1-4302-3115-8.

**Lecture No.27**

**Validating user’s input**

Once the data is received we can use PHP to validate the user’s input. In PHP, usually we declare a regular expression and the use a PHP function (preg\_match()) to compare user’s input with the regular expression.

1. **Validating user’s input:**

A regular expression is a concise notation to describe patterns in strings. Regular expressions provide the foundation for describing or matching data according to defined syntax rules. We can use regular expressions to define a pattern and check to see if it can be applied to your data.

**Regular Expression Syntax:** A regular expression is nothing more than a pattern of characters itself, matched against a certain input text. The structure of a regular expression is similar to that of a typical arithmetic expression. It means, various elements or operators are combined to form a more complex expression. Following are the most basic notations used to define a regular expression

* ^: match strings that start with the given pattern
* $: match strings that end with the given pattern
* -: means a range of characters
* [ ]: makes a class of characters
* [^ ]: negates the class of character
* ?: matches the character, class or sub-pattern 0 or 1 time equal to {0,1}
* +: matches the character, class or sub-pattern 1 or more times equals to {1, }
* \*: matches the character, class or sub-pattern 0 or any number of time equals {0, }
* \d: means exactly as [0-9]
* \D: means exactly as [^0-9]
* \w: means exactly as [a-zA-Z0-9]

**RE examples:** in the following examples, we write some regular expressions to validate user’s input

**Validating date:** dates are the most common input we need to take from a user. A date consists of three parts: day, month and year. These parts are usually separated by – or / symbol. To validate a date we can use the following regular expression. This expression consists of three parts. In each part the allowed character’s range is given \d which means exactly the same as [0-9] and are separated by – symbol.

* + **|^\d{2}-\d{2}-\d{4}$|**

**Validating CNIC:** In the following example, we have declared a regular expression to validate CNIC number. A CNIC number consists of three parts. The first part consists of five digits, then a – symbol. The second part consists of seven digits and then again a – symbol; and then one digit.

* + **|^\d{5}-\d{7}-\d{1}$|**

**Validating Email:** The following example validates an email address.

* + **|^[a-zA-Z0-9\_.]+@[a-z]{3,5}.[a-z]{2,3}$|**

**Evaluating user’s input:** After declaring a regular expression, we have to match the user’s input.The preg\_match() function searches a string for a specific pattern, returning TRUE if it exists and FALSE otherwise. We can use the following syntax to use this function

int preg\_match(“pattern”,$string);

In the following example we have validated user’s input. First we have declared a form and in the action page we have validated user’s email.

|  |  |
| --- | --- |
| <body>  <form method=“post” action=”action.php”>  <input type=“text” name=“name”>  <input type=“text” name=“email”>  <input type=“submit”>  </form>  </body> | **action.php**  <?php  $name = $\_POST[‘name’];  $email = $\_POST[‘email’];  if(!preg\_match("|^[a-zA-Z0-9\_.]+@[a-z]{3,5}.[a-z]{3}$|",$email))  echo "Invalid Email address";  ?> |

1. **String functions in PHP:**

In addition to regular expressions, PHP provides many string functions which can also be used to evaluate user’s input. Some of the basic string functions are discussed here

**strlen():** strlen() function is used to find the length of the string. Following is the syntax of a regular expression; int strlen($string);

**strcmp():** This function is used to compare two strings. It returns 0 if strings are equal, 1 if first string is greater and -1 if second is greater. The syntax of the strcmp() function is

* + int strcmp($string1,$string2);

**strcasecmp():** strcasecmp() function compares two strings in case insensitive manner.

* + strcasecmp($string1,$string2);

**strtolower():** It scnverts a string in lower case.

* + strtolower($string);

**strtoupper():** It converts a string in upper case

* + strtoupper($string);

**ucfirst():** This function converts the first character of a string to upper case.

* + ucfirst($string);

**ucwords():** It converts the first character of each word in a string to upper case

* + ucfirst($string);

**strpos():** It finds the position of the first case-sensitive occurrence of a substring in a string

* + strpos($string,sub-string);

**strrpos():** It finds the position of the last case-sensitive occurrence of a substring in a string

* + strrpos($string,sub-string);

**substr\_count():** It returns the number of times one string occurs within another

* + substr\_count($string,sub-string);

**Example:** In the following example we have taken user’s name, password and re-type password as input. On the action page, we have used string functions to evaluate user’s input. First we find the length of the password using strlen() function and if password’s length is less than six then we have displayed an error message. Similarly, we have compared password and re-type password and if they don’t match; an error message is displayed. Then we have used strtolower(), strtoupper(), ucfirst() and ucword() functions on user’s name. Similarly, we have applied strops(), strrpos() and substr\_count() functions on user’s name.

|  |  |
| --- | --- |
| <html>  <head>  <title>String Function</title>  </head>  <body>  <form id="form1" name="form1" method="post" action="str\_action.php">  Name: <input type="text" name="name" />  Type new password:  <input type="text" name="pass" />  Re-type new password:  <input type="text" name="pass1" />  <input type="submit" name="Submit value="Submit" />  </form>  </body>  </html> | <?php  $name = $\_POST['name'];  $pass = $\_POST['pass'];  $pass1 = $\_POST['pass1'];  if(strlen($pass)<6)  echo "Too short password";  if(strcmp($pass,$pass1)<>0)  echo "Password mismatch";  echo strtolower($name)."<br>";  echo strtoupper($name)."<br>";  echo ucfirst($name)."<br>";  echo ucwords($name)."<br>";  echo strpos($name,'a')."<br>";  echo strrpos($name,'a')."<br>";  echo substr\_count($name,'a')."<br>";  ?> |

**References:**

* **Chapter 9, “**Beginning PHP and MySQL” by W. Jason Gilmore, Apress publisher, 4th edition; 2010, ISBN-13 (electronic): 978-1-4302-3115-8.

**Lecture No.28**

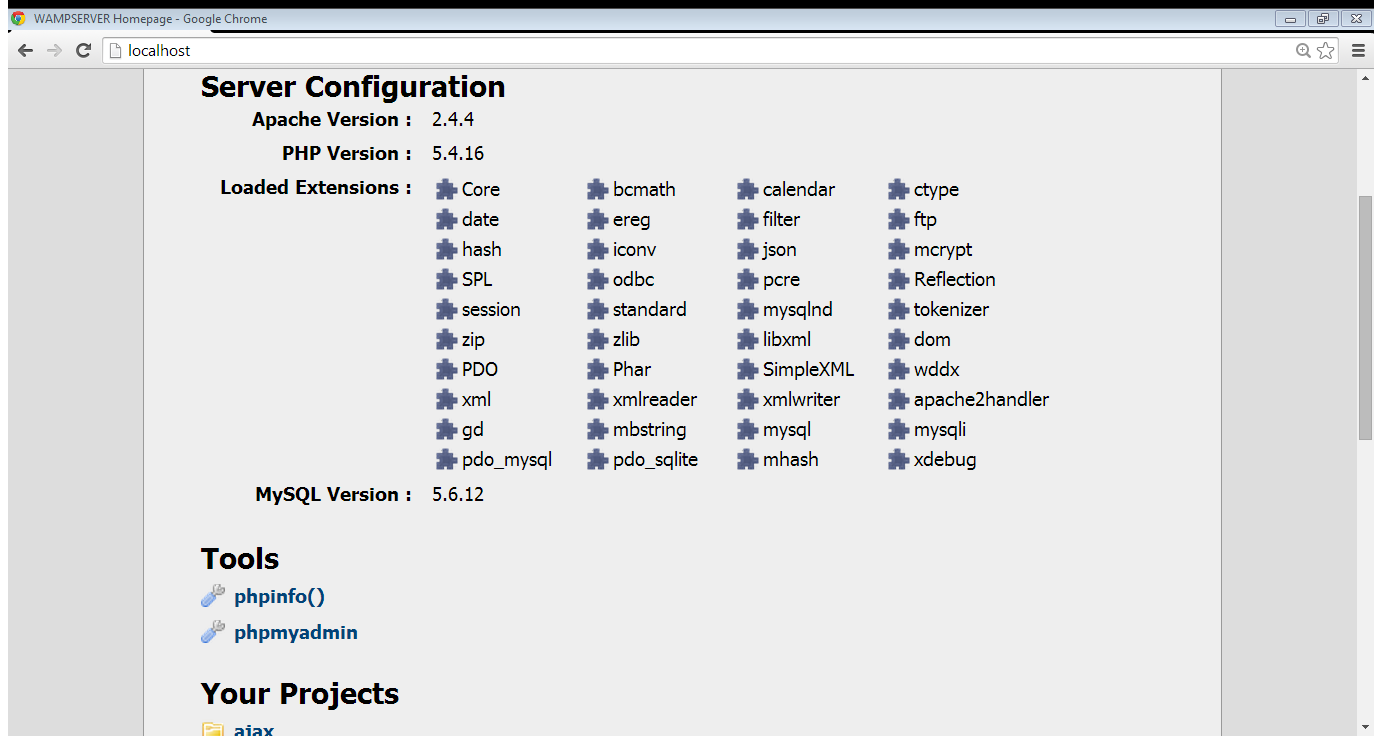
**Connecting PHP with MySQL**

Once we have evaluated the user’s input, now it is time to process this information. This information can be used to work with databases. For example, we can store this information to database or we can search some information from database etc. To work with database we have to connect our PHP script with database and then, we write SQL instructions. In this lecture we will learn how we can connect PHP page with database and how to insert data into database tables.

1. **Creating database in MySQL using WAMP server:**

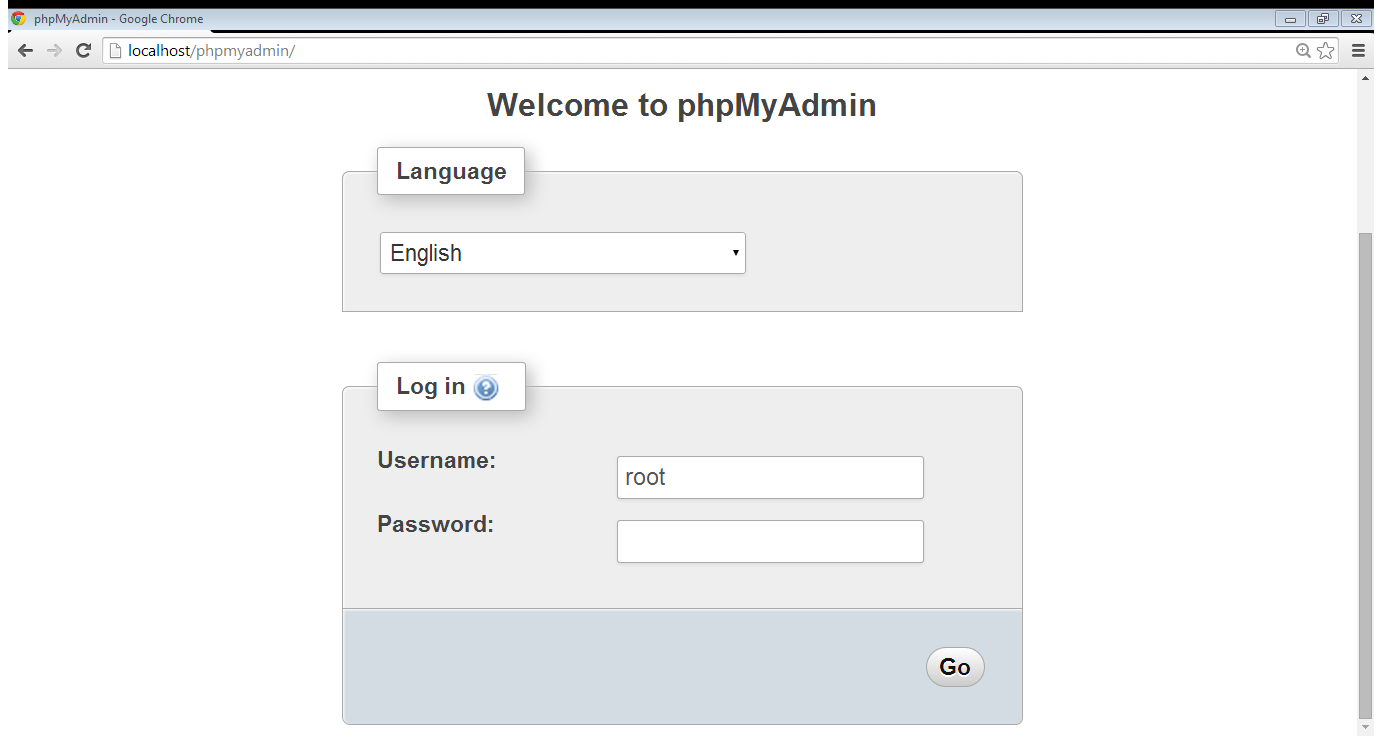
To work with database, we have to create database. We can write SQL instructions to create a database. While using WAMP, we can create database using button based menus. In this section we discuss, how we can create a database in WAMP server.

**Go to home page of WAMP server:** To create a database, we have to PHP MyAdmin. We can find the link of PHP MyAdmin at WAMP server’s home page. The following picture shows the location of this link.



**Select phpmyadmin**

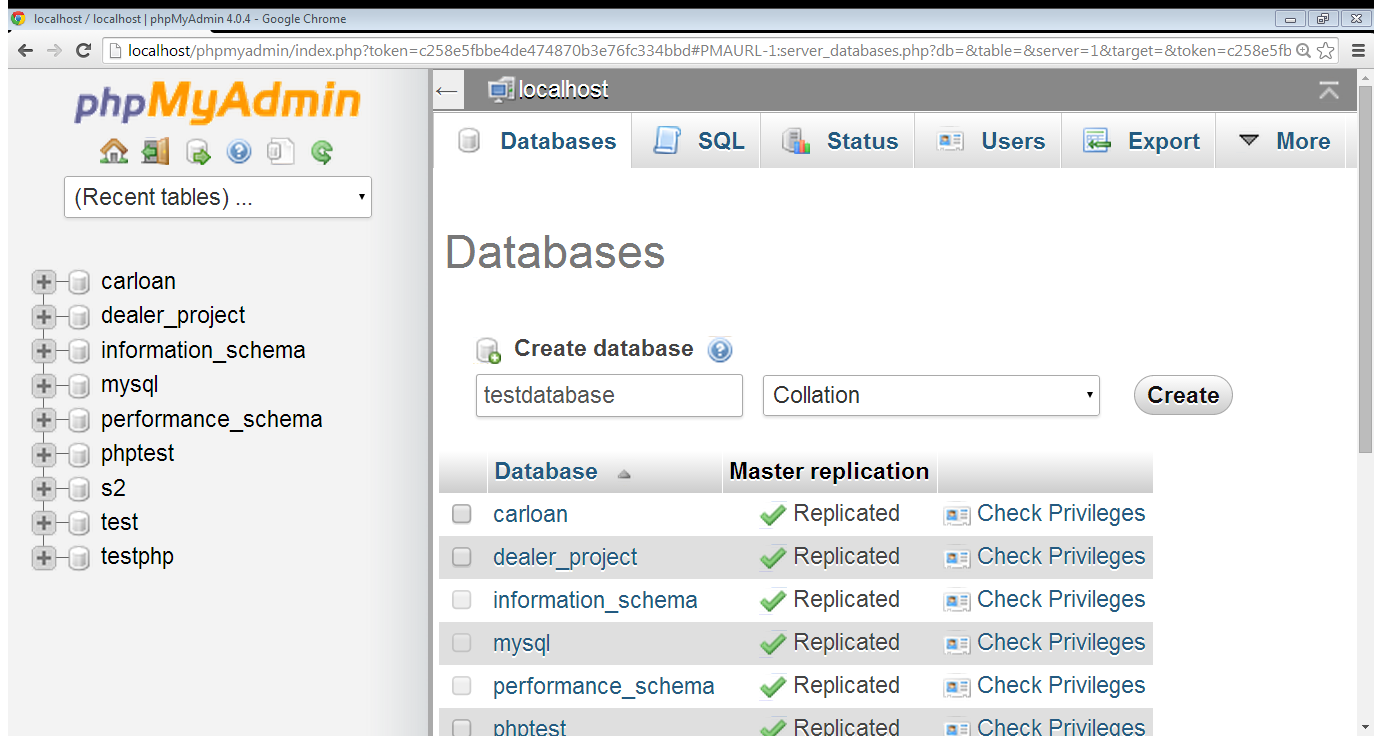
**Login to MySQL:** at phpmyadmin’s home page, we can login to MySQL. We can use an already created user root with empty password.



**User name is root**

**Empty password**

**Enter database name and click create database button:** After logging in to MySQL, we have to enter database name and select create database link. In the following figure we have created a database with the name testdatabase.

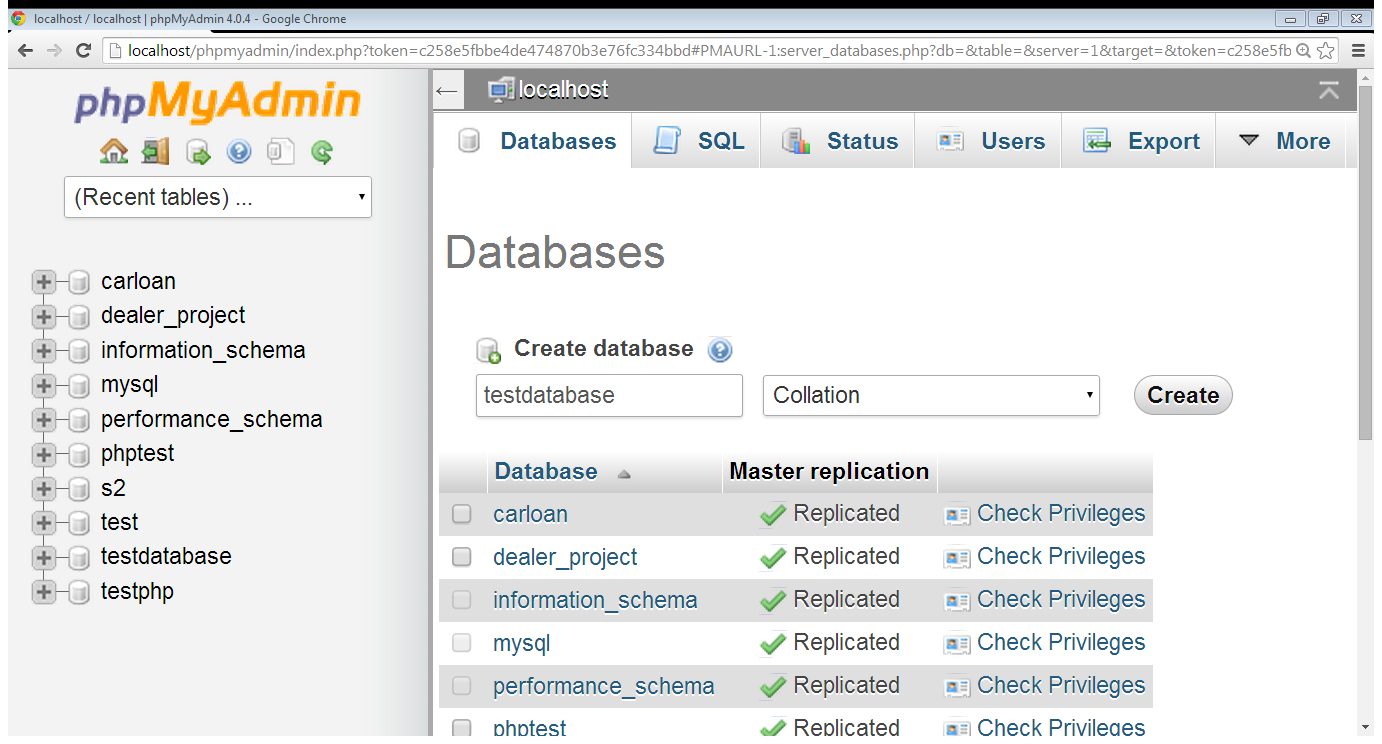


**Select create**

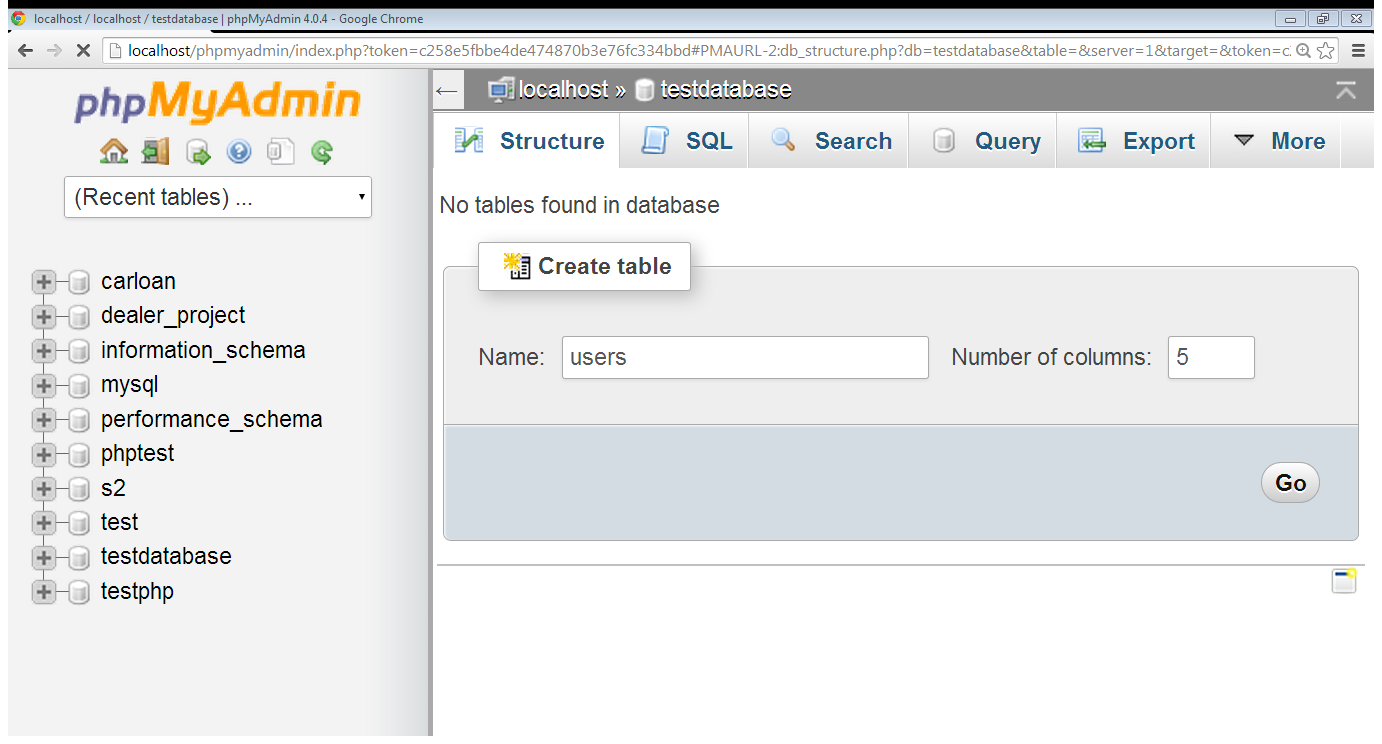
**Enter database name**

**Select create**

**1.1 Creating table in database:** after creating a database, we have to create tables in that database to store information. We follow the following steps to create a table in a database in WAMP.

**Select the database:** first of all, we select the database where we have to create the table as shown in the following figure that we select the testdatabase.

**Select database**

**Table name:** after selecting the database, we have to enter table name and number of columns; and click go button to create a table. In the following figure, we have created a table ‘users’ with five columns.

**Click go**

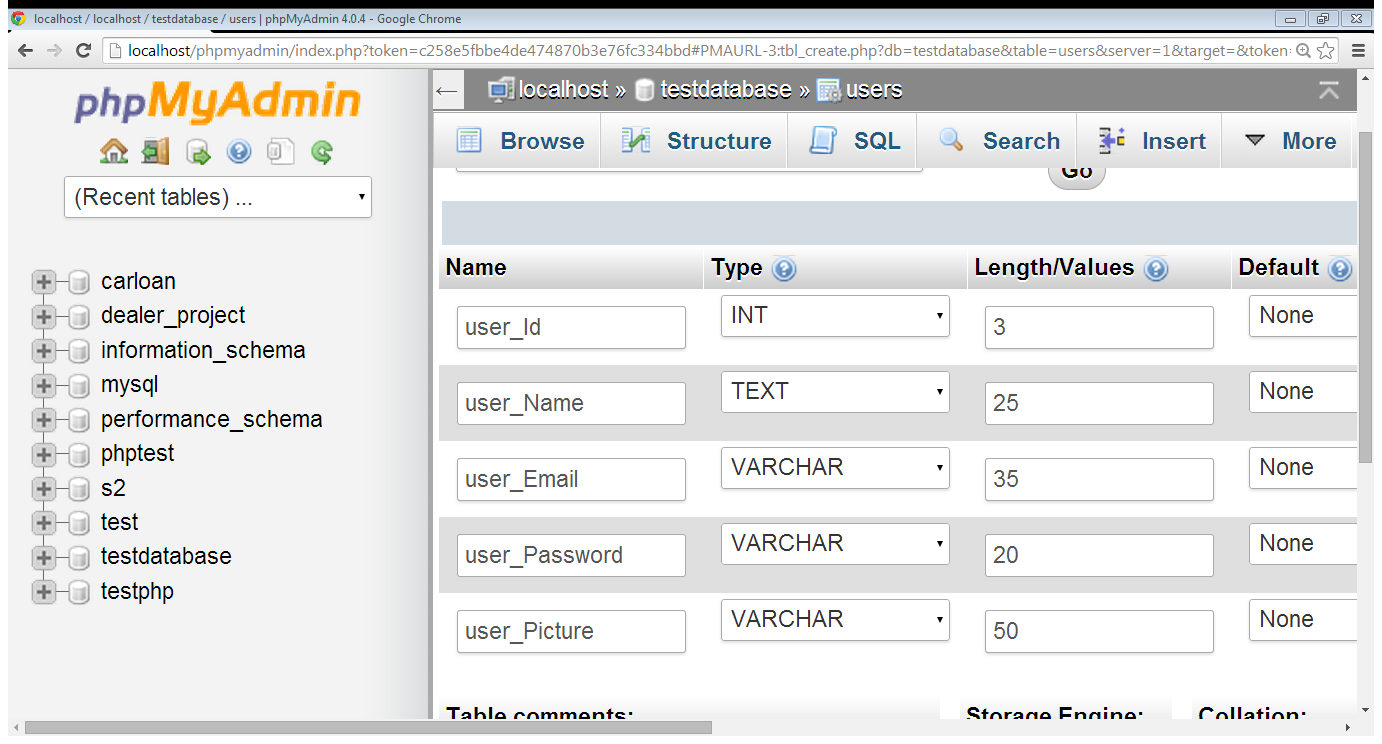
**Enter no. of columns**

**Table name**

**Enter no. of columns**

**Click go**

**Table structure:** after create a table, we define the table structure. We enter the name of the columns and their attributes; and click at the save button to create the table.



**Max length**

**Data-type**

**Column name**

1. **Database Connection:**

To connect a web page with MySQL, first of all we have to make a connection with the database server and then we select the database. PHP provides a function mysql\_connect() to make the connection with database server. The syntax of this function is

mysql\_connect(‘host-name’,’user’,’password’)

For example, to connect to the server using root ‘user’ and localhost we write

mysql\_connect(‘localhost’,’root’,’’);

As we are connected with the database server, now we have to select the database. For this we can use mysql\_select\_db() function. The syntax of this function is

mysql\_select\_db(‘database-name’)

If we use the ‘testdatabse’ database created earlier, we can select it by

mysql\_select\_db(‘testdatabse’)

In the following example we write the code to connect to MySQL using PHP

**<?php**

**mysql\_connect(‘localhost’,’root’,’’);**

**mysql\_select\_db(‘testdatabase’);**

**?>**

1. **Inserting data in MySQL table using PHP:**

Databases are stores of information. They allow one to easily record and then access large amounts of information for a wide variety of purposes. Data insertion, deletion, updating and searching are the most basic functions, we perform on data. To insert data in a table we follow these steps

* make connection with database
* prepare the insert query
* execute the query

In previous section we discussed how we can make connection with database; in this section we will learn how we can insert data in a MySQL table using PHP. First we look at the syntax of insert query in MySQL.

**INSERT INTO `table\_name`**

**(list of columns)**

**VALUES (list of values)**

For example, to insert a record in the ‘users’ table created in the ‘testdatabase’ we can write the SQL query as

**INSERT INTO users**

**(‘user\_Name’,’user\_Email’,’user\_Password’)**

**VALUES (‘ali’,’ali@yahoo.com’,’123’)**

After preparing insert query, we have to execute this query. For this we use mysql\_query() function. The syntax of the function is

**mysql\_query(‘sql-instruction’);**

In the following example we created a form which takes user’s name, email and password as input while its action page inserts the record in the database.

|  |  |
| --- | --- |
| <html>  <head>  <title>User Registration Form</title>  </head>  <body>  <form id="form1" name="form1" method="post" action="reg\_action.php">  Name: <input type="text" name="name" />  <br>  Email: <input type="text" name="email" />  Password:  <input type="text" name="password" /> <input type="submit" name="Submit" value="Register" />  </form>  </body>  </html> | <?php  $name = $\_POST['name'];  $email = $\_POST['email'];  $password = $\_POST['password'];  mysql\_connect("localhost","root",""); mysql\_select\_db("testdatabase");  $sql="INSERT INTO `testdatabase`.`users` (`user\_Name`, `user\_Email`, `user\_Password`) VALUES ('$name', '$email', '$password')";  mysql\_query($sql);  ?> |

**References:**

* Chapter 30, “Beginning PHP and MySQL” by W. Jason Gilmore, Apress publisher, 4th edition; 2010, ISBN-13 (electronic): 978-1-4302-3115-8.

**Lecture No.29**

**Uploading files using PHP**

In the previous lecture, we discussed how we can connect a web page with MySQL using PHP. We also discussed the data insertion using PHP. In this lecture we will learn how we can upload files to web server in PHP.

1. **File uploading in PHP:**

Sometimes we need to allow a user to attach some file with the form as input. For example, while registering a user we can ask the user to attach its picture. The attached file exists at the client’s machine. To use this attached file we need to have it at the server. For this, the attached file is transferred from the client’s machine to the server. This process is called file uploading.

**$\_FILES super global variable:** when a user submits a form with an attachment, information about the attached file is stored in $\_FILES super global variable. $\_FILES super global variable contains any item uploaded to the server when the post method is used. Like other super global variables, $\_FILES is also an array type variable and it is created automatically. We can access its contents on other pages. $\_FILES keeps information about the name, size and type of the attached file.

To allow a user to attach a file with a form, we have to use the post method and set the enctype attribute of the form tag. The enctype attribute specifies how the form-data should be encoded when submitting it to the server. The default value of enctype attribute is ‘application/x-www-form-urlencoded’. It means all data is encoded before sending to the server. When we have to allow a user to attach a file with the form, we don’t encode the data. Therefore, we have to set the value of enctype attribute to ‘multipart/form-data’; it means no encoding is performed before sending the data to the server.

**Accessing file related information:** as we discussed, $\_FILES super global array contains information about the attached files. We can retrieve this information on the action page. Suppose in a form; we have a field of file type with name ‘pic’ where the user can attach its picture. When the user submits the form, a field in #\_FILES super global variable is created. The name of the filed will be same as of the input filed in the form that is ‘pic’ in this example. We can retrieve the name, size and type of the file from $\_FILES super global array as

* $\_FILES[‘pic’][‘name’];
* $\_FILES[‘pic’][‘type’];
* $\_FILES[‘pic’][‘size’];

In the following example we create a form where a user can attach its picture. On the action page we retrieve the information about the user’s attached file.

|  |  |
| --- | --- |
| <html>  <head>  <title>File Uploading</title>  </head>  <body>  <form id="form1" name="form1" enctype="multipart/form-data" method="post" action="fileaction.php">  Please attach your picture:  <input type="file" name="pic" />  <input type="submit" name="Submit" value="Submit" />  </form>  </body>  </html> | **fileacton.php**  <?php  echo $filename=$\_FILES['pic']['name'];  echo $size =$\_FILES['pic']['size'];  echo $type =$\_FILES['pic']['type'];  ?> |

**Uploading file:** in previous example, we retrieved the information about the user’s attached file on the action page. To upload this file to the server we use the PHP function move\_uploadded\_file(). The syntax of the function is

**bool move\_uploaded\_file ( string $filename , string $destination );**

This function checks to ensure that the file designated by filename is a valid upload file (meaning that it was uploaded via PHP's HTTP POST upload mechanism). If the file is valid, it will be moved to the filename given by destination.

We can write the file uploading script as

* First we identify the file to be uploaded. For this we use tmp\_name of the attached file
* We define the destination where the file is to be uploaded. This destination consists of the name of the directory and the file name
* Upload the file using move\_uploaded\_file() function

Following example code uploads the user’s attached file to the ‘uploads’ directory on the server

|  |  |
| --- | --- |
| <html>  <head>  <title>File Uploading</title>  </head>  <body>  <form id="form1" name="form1" enctype="multipart/form-data" method="post" action="fileaction.php">  Please attach your picture:  <input type="file" name="pic" />  <input type="submit" name="Submit" value="Submit" />  </form>  </body>  </html> | **fileacton.php**  <?php  $filename = $\_FILES['pic']['name'];  $size = $\_FILES['pic']['size'];  $type = $\_FILES['pic']['type'];  $tmpname= $\_FILES['pic']['tmp\_name'];  $destination="uploads/"$filename;  move\_uploaded\_file($tmpname,$destination);  ?> |

**Applying restrictions on attached files:** We can impose different restriction on attached files. The most common restrictions are size of the file and the type of the file. In the following example we check the file size and type before uploading the file

|  |  |
| --- | --- |
| <html>  <head>  <title>File Uploading</title>  </head>  <body>  <form id="form1" name="form1" enctype="multipart/form-data" method="post" action="fileaction.php">  Please attach your picture:  <input type="file" name="pic" />  <input type="submit" name="Submit" value="Submit" />  </form>  </body>  </html> | **fileacton.php**  <?php  $filename = $\_FILES['pic']['name'];  $size = $\_FILES['pic']['size'];  $type = $\_FILES['pic']['type'];  $tmpname= $\_FILES['pic']['tmp\_name'];  $destination="uploads/"$filename;  if($size<100000 AND $type=='image/jpeg')  move\_uploaded\_file($tmpname,$destination);  else  echo "Invalid File";  ?> |

**File renaming:** Sometimes users attach files with same name. When such a file is uploaded to the server the previous file is overwritten. To avoid such scenario we have to rename the file before uploading. In the following example we create a random umber and attach this number with the file name before uploading the file

|  |  |
| --- | --- |
| <html>  <head>  <title>File Uploading</title>  </head>  <body>  <form id="form1" name="form1" enctype="multipart/form-data" method="post" action="fileaction.php">  Please attach your picture:  <input type="file" name="pic" />  <input type="submit" name="Submit" value="Submit" />  </form>  </body>  </html> | **fileacton.php**  <?php  $filename = $\_FILES['pic']['name'];  $size = $\_FILES['pic']['size'];  $type = $\_FILES['pic']['type'];  $tmpname = $\_FILES['pic']['tmp\_name'];  $destination="uploads/".rand().$filename;  if($size<100000 AND $type=='image/jpeg')  move\_uploaded\_file($tmpname,$destination);  else  echo "Invalid File";  ?> |

**Storing file reference to database:** in the previous section we learnt file uploading. In web applications, we have to use these uploaded files. For this we have to store reference of the uploaded file in the database. In the above example, $destination stores the information about the name and location of the file. We can store this variable in the database as a reference.

In the following example we create a form which asks the users to enter their name, email and password; and to attach their picture. On the action page we upload the attached file and store the user’s information in the ‘users’ table in the ‘testdatabase’ database.

|  |  |
| --- | --- |
| <html>  <head>  <title>User Registration Form</title>  </head>  <body>  <form enctype="multipart/form-data" id="form1" name="form1" method="post" action="reg\_action.php">  Name:  <input type="text" name="name" />  Email:  <input type="text" name="email" />  Password:  <input type="text" name="password" />  Your Picture:  <input type="file" name="picture" />  <input type="submit" name="Submit" value="Register" />  </form>  </body>  </html> | **reg\_action.php**  <?php  $name = $\_POST['name'];  $email = $\_POST['email'];  $password = $\_POST['password'];  $filename =$\_FILES['picture']['name'];  $size =$\_FILES['picture']['size'];  $type =$\_FILES['picture']['type'];  $tmpname = $\_FILES['picture']['tmp\_name'];  $destination= "uploads/".rand().$filename;  move\_uploaded\_file($tmpname,$destination);  mysql\_connect("localhost","root",""); mysql\_select\_db("testdatabase");  $sql="INSERT INTO `testdatabase`.`users` (`user\_Name`, `user\_Email`, `user\_Password`, `user\_Picture`)  VALUES ('$name', '$email', '$password', '$destination')";  mysql\_query($sql);  header("location:form.php?msg=Recod is added");  ?> |

**Reference:**

* **Chapter 15 and 30, “**Beginning PHP and MySQL” by W. Jason Gilmore, Apress publisher, 4th edition; 2010, ISBN-13 (electronic): 978-1-4302-3115-8.

**Lecture No.30**

**Retrieving data from MySQL using PHP**

In previous lecture we discussed one of the basic tasks we perform while working with database that is data insertion. We have to retrieve this data in order to see/display the stored information. In this lecture we will learn how to retrieve data from MySQL using PHP.

1. **Retrieving data from MySQL using PHP**

Once data is stored in the database, we can retrieve and process this data. Data retrieval is one of the most important functions. We can write PHP script to read data from database. We follow the following steps to read and display data on the web page using PHP.

**Connection with database:** The basic requirement to retrieve data from the database is that we have a connection. In previous lectures, we have already discussed how we can connect a web page with database. The following example shows the database connection script. Here, we have used the database ‘tesstdatabase’ created in lecture 28.

**<?php**

**mysql\_connect(‘localhost’,’root’,’’);**

**mysql\_select\_db(‘testdatabase’);**

**?>**

**Selecting data:** once we have connected with database, now we have to select the data we want to read. For selecting the data we use the SQL select instruction. The syntax of select instruction is given below

**SELECT column-name**

**FROM table-name**

For example if we have to select user’s name from the users table, we can do this by

**SELECT user\_Name**

**FROM users**

In the above example we have selected only one column, If we have to select all of the column we can do this by

**SELECT \***

**FROM users**

Similarly we can select data on the basis of some conditions. For this we use ‘where’ clause in the select instruction. For example if we want to select the users whose id is greater than four, we write the following select instruction

**SELECT \***

**FROM users**

**WHERE user\_Id>4**

After writing the select instruction, we have to execute this instruction. In the following example we selected user’s record from users table and assign the result to a variable

**<?php**

**mysql\_connect(‘localhost’,’root’,’’);**

**mysql\_select\_db(‘testdatabase’);**

**$sql = ‘select \* from users’;**

**$result = mysql\_query($sql);**

**?>**

We can use mysql\_num\_rows() function to count the number of rows selected by a select instructions. In the following example we have used this function to count the number of users exist in the database

**<?php**

**mysql\_connect(‘localhost’,’root’,’’);**

**mysql\_select\_db(‘testdatabase’);**

**$sql = ‘select \* from users’;**

**$result = mysql\_query($sql);**

**$users = mysql\_num\_rows($result);**

**echo “There are total ”. $users .”users found”;**

**?>**

**Data fetching:** in the previous example we have selected the data from a table and assigned this data to a variable. This variable keeps all of the selected rows. We can use mysql\_fetch\_array() to fetch one record or row. mysql\_fetch\_array() function draws one row and return this row in the form of an array. Values of this array can be accessed either by index or by name. The name of the column in the table becomes the name of that value in the array. After fetching the array, we can process the fetched data. In the following example we have fetched and displayed a record from the $result of the previous example

**<?php**

**mysql\_connect(‘localhost’,’root’,’’);**

**mysql\_select\_db(‘testdatabase’);**

**$sql = ‘select \* from users’;**

**$result = mysql\_query($sql);**

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

**echo “User name is”.$row[1];**

**echo “User email is”.$row[2];**

**?>**

In the following example, we have summarized the lecture. We have displayed users from the users table of testdatabase in tabular form.

<html>

<head>

<title> Retrieving data</title>

</head>

<body>

<?php

mysql\_connect(‘localhost’,’root’,’’);

mysql\_select\_db(‘testdatabase’);

$sql = "select \* from users";

$result = mysql\_query($sql);

$users= mysql\_num\_rows($result);

?>

<table border="1">

<tr bgcolor="#CCCC00">

<th>User Name</th>

<th>User Email</th>

<th>User Password</th>

<th>User Picture</th>

</tr>

<?php

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

{

?>

<tr bgcolor="#CCCCCC">

<td><?php echo $rows[1]; ?> </td>

<td><?php echo $rows[2]; ?> </td>

<td><?php echo $rows[3]; ?> </td>

<td><img src="<?php echo $rows[4]; ?>" height="100" width="100"> </td>

</tr>

<?php

}

?>

</table>

</body>

</html>

**References:**

* Chapter 30, “Beginning PHP and MySQL” by W. Jason Gilmore, Apress publisher, 4th edition; 2010, ISBN-13 (electronic): 978-1-4302-3115-8.

**Lecture No.31**

**Deleting and updating records**

In previous lectures, we discussed some basic function which we perform on data including data insertion and data selection. In this lecture we will learn how we can delete data from database and how we can update the existing data.

1. **Deleting data:**

Deletion is the process of removing data from database. When we feel that certain data is no more required, we remove that data in-order to reduce the size of the database. SQL provides the delete instruction to delete records from the database. The syntax of the delete instruction is

**DELETE FROM** table-name

**WHERE** condition

Here, DELETE FROM is the keyword while table-name is the name of the table from which we want to delete data. WHERE is also a keyword and here, we give the condition about that which record is to be deleted. For example the following code deletes the record of the user whose id is 5 from the users table (we used in previous lecture)

DELETE FROM users

WHERE user\_Id =5

We can delete data from the database using PHP. For this, first of all we make a connection to the database and then we execute the delete instruction. The following example shows the PHP scripts that deletes the record of the user whose id is 5 from the **‘users’** table of the **’testdatabase’** database.

<?php

mysql\_connect(‘localhost’,’root’,’’);

mysql\_select\_db(‘testdatabase’);

$sql=“DELETE FROM users

WHERE user\_Id=5”;

mysql\_query($sql);

?>

**Sending data with link:** while studying HTML, we discussed how we can link documents with each other. Sometimes we have to send some data with a link. We can attach data with a link as

<a href=”page-reference?data-reference=”value””> …….. </a>

The attached data is sent along with the URL as a query string parameter. As we know, in PHP $\_GET super global array keeps the data which is attached with the URL. Therefore we can get this data on the next page. In the above line of coed, the page reference is the reference of the page where we want to move, data-reference is the name of the field of $\_GET array where the data is stored and value is the actual data. On next page we can get this value by $\_GET[‘data-reference’].

In the following example we display the record of the users from the ‘users’ table (as we did in the previous lecture) and add a new column which links to a page ’delete.php’. With this link we attach the id of the user. The delete.php gets the id of the user from the link and deletes that record.

|  |  |
| --- | --- |
| <html>  <head>  <title> Retrieving data</title>  </head>  <body>  <?php  mysql\_connect(‘localhost’,’root’,’’);  mysql\_select\_db(‘testdatabase’);  $sql = "select \* from users";  $result = mysql\_query($sql);  ?>  <table border="1">  <tr bgcolor="#CCCC00">  <th>User Name</th>  <th>User Email</th>  <th>User Password</th>  <th>User Picture</th>  <th>Action</th>  </tr>  <?php  while($rows=mysql\_fetch\_array($result))  {  ?>  <tr bgcolor="#CCCCCC">  <td><?php echo $rows[1]; ?> </td>  <td><?php echo $rows[2]; ?> </td>  <td><?php echo $rows[3]; ?> </td>  <td><img src="<?php echo $rows[4]; ?>" height="100" width="100"> </td>  <td>  <a href="delete.php?id=<?php echo $rows[0];?>" onClick="return confirm('Delete This account?')"> Delete</a></td>  </tr>  <?php  }  ?>  </table>  </body>  </html> | <?php  $id = $\_GET['id'];  include('connection.php');  $sql="delete from users  where user\_Id=$id";  mysql\_query($sql);  ?> |

1. **Updating the record:**

Updating the data is also an important function, we perform on data. SQL provides us the update instruction to update some existing data. The syntax of the update instruction is

**UPDATE** table-name

**SET** column-names = values

**WHERE** condition

In this instruction UPDATE is a keyword, table-name is the name of the table where we want to update some data, SET is also a keyword, column-names are the names of the columns in the table, values are new values for columns, WHERE is a keyword; and condition is the condition about the record to be updated. In the following example we update a record whose id is 1 in the users table

UPDATE users

SET user\_Name = ‘Ali’,

user\_Email = ‘ali@yahoo.com’,

user\_Password=‘123’

WHERE user\_Id=1

To update a record using PHP, first of all we make a connection with database and then execute the update instruction as shown below

<?php

mysql\_connect(‘localhost’,’root’,’’);

mysql\_select\_db(‘testdatabase’);

$sql =“UPDATE users

SET user\_Name = ‘Ali’,

user\_Email = ‘ali@yahoo.com’,

user\_Password=‘123

Where user\_Id=1’’;

mysql\_query($sql);

?>

**Reference:**

**Chapter 30, “**Beginning PHP and MySQL” by W. Jason Gilmore, Apress publisher, 4th edition; 2010, ISBN-13 (electronic): 978-1-4302-3115-8.