69/B Monwara Plaza (Level-5)
East Panthapath
Dhaka-1205
Cell: 01713 432003

Lecture Sheet on “PHP+MySQL”
Day-11: (PHP+MySQL)

Objectives

- ❖ Describe how arguments and PHP functions relate to the database
- ❖ Understand the importance of associative and indexed arrays.
- ❖ Update database records using PHP scripts
- ❖ Upload/Display of Binary files to MySQL database.

Topics

- ❖ Connecting to MySQL and selecting the database
- ❖ Executing simple queries
- ❖ Retrieving query results
- ❖ Ensuring secure SQL
- ❖ Counting returned records
- ❖ Updating records with PHP
- ❖ MySQL results to Excel File
- ❖ Freeing memory
- ❖ Locking of Tables

Get Data From MySQL Database

Using PHP you can run a MySQL SELECT query to fetch the data out of the database. You have several options in fetching information from MySQL. PHP provide several functions for this. The first one is mysql_fetch_array()which fetch a result row as an associative array, a numeric array, or both.

Below is an example of fetching data from MySQL, the table contact have three columns, name, subject and message.

```
<?php
$query  = "SELECT name, subject, message FROM contact";
$result = mysql_query($query);

while($row = mysql_fetch_array($result, MYSQL_ASSOC))
{
   echo "Name :{$row['name']} <br>" .
        "Subject : {$row['subject']} <br>" .
        "Message : {$row['message']} <br><br>";
}
?>
```

The while() loop will keep fetching new rows until mysql_fetch_array() returns FALSE, which means there are no more rows to fetch. The content of the rows are assigned to the variable $row and the values in row are then printed. Always remember to put curly brackets when you want to insert an array value directly into a string.

In above example I use the constant MYSQL_ASSOC as the second argument to mysql_fetch_array(), so that it returns the row as an associative array. With an associative array you can access the field by using their name instead of using the index. I think it's more informative to use $row['subject'] instead of $row[1].

PHP also provide a function called mysql_fetch_assoc() which also return the row as an associative array.

```
<?php
$query  = "SELECT name, subject, message FROM contact";
$result = mysql_query($query);

while($row = mysql_fetch_assoc($result))
{
   echo "Name :{$row['name']} <br>" .
        "Subject : {$row['subject']} <br>" .
        "Message : {$row['message']} <br><br>";
}
?>
```

You can also use the constant MYSQL_NUM, as the second argument to mysql_fetch_array(). This will cause the function to return an array with numeric index.

```
<?php
$query  = "SELECT name, subject, message FROM contact";
$result = mysql_query($query);

while($row = mysql_fetch_array($result, MYSQL_NUM))
{
echo "Name :{$row[0]} <br>"."Subject : {$row[1]} <br>"."Message : {$row[2]} <br><br>";
}
?>
```

Using the constant MYSQL_NUM with mysql_fetch_array() gives the same result as the function mysql_fetch_row().

There is another method for you to get the values from a row. You can use list(), to assign a list of variables in one operation.

```
<?php
$query  = "SELECT name, subject, message FROM contact";
$result = mysql_query($query);

while(list($name,$subject,$message)= mysql_fetch_row($result))
{
   echo "Name :$name <br>" ."Subject : $subject <br>" ."Message : $message <br><br>";
}
?>
```

In above example, list() assign the values in the array returned by mysql_fetch_row() into the variable $name, $subject and $message. Of course you can also do it like this

```
<?php
$query  = "SELECT name, subject, message FROM contact";
$result = mysql_query($query);
while($row = mysql_fetch_row($result))
{
   $name    = $row[0];
   $subject = $row[1];
   $message = $row[2];
   echo "Name :$name <br>" ."Subject : $subject <br>" ."Message : $row <br><br>";
}
?>
```

So you see you have lots of choices in fetching information from a database. Just choose the one appropriate for your program

Freeing the memory?

In some cases a query can return large result sets. As this results are stored in memory there's a concern about memory usage. However you do not need to worry that you will have to call this function in all your script to prevent memory congestion. In PHP all results memory is automatically freed at the end of the script's execution.

But you are really concerned about how much memory is being used for queries that return large result sets you can use mysql_free_result(). Calling this function will free all memory associated with the result identifier ($result).

Using the above example you can call mysql_free_result() like this :

```
<?php
$query  = "SELECT name, subject, message FROM contact";
$result = mysql_query($query);

while($row = mysql_fetch_row($result))
{
...
}
mysql_free_result($result);
?>
```

Convert MySQL Query Result To Excel

Using PHP to convert MySQL query result to Excel format is also common especially in web based finance applications. The finance data stored in database are downloaded as Excel file for easy viewing. There is no special function in PHP to do the job. But you can do it easily by formatting the query result as tab separated values or put the value in an HTML table. After that set the content type to application/vnd.ms-excel

```
<?php
$query  = "SELECT fname, lname FROM students";
$result = mysql_query($query) or die('Error, query failed');

$tsv  = array();
$html = array();
while($row = mysql_fetch_array($result, MYSQL_NUM))
{
  $tsv[]  = implode("\t", $row);
  $html[] = "<tr><td>" .implode("</td><td>", $row) .           "</td></tr>";
}
```

```
$tsv = implode("\r\n", $tsv);
$html = "<table>" . implode("\r\n", $html) . "</table>";

$fileName = 'mysql-to-excel.xls';
header("Content-type: application/vnd.ms-excel");
header("Content-Disposition: attachment; filename=$fileName");

echo $tsv;
//echo $html;   when you print the $html you dont have to set the headers!

?>
```

In the above example $tsv is a string containing tab separated values and $html contain an HTML table. I use implode() to join the values of $row with tab to create a tab separated string.

After the while loop implode() is used once again to join the rows using newline characters. The headers are set and the value of $tsv is then printed. This will force the browser to save the file as mysql-to-excel.xsl

Try running the script in your own computer then try commenting echo $tsv and uncomment echo $html to see the difference.

MySQL Update and Delete

There are no special ways in PHP to perform update and delete on MySQL database. You still use mysql_query() to execute the UPDATE or DELETE statement.

For instance to update a password in mysql table for username phpcake can be done by executing an UPDATE statement with mysql_query() like this:

```
php?>
mysql_select_db('mysql')or die('Error, cannot select mysql database');

$query = "UPDATE user SET password = PASSWORD('newpass')". "WHERE user = 'phpcake'";

mysql_query($query) or die('Error, query failed');
?>
```

There is one important thing that you should be aware of when updating and deleting rows from database. That is **data integrity**.

If you're using **InnoDB** tables you can leave the work of maintaining data integrity to MySQL . However when you're using other kind of tables you need to enforce the data integrity manually.

To make sure that your update and delete queries will not break the data integrity. You have to make appropriate update and delete queries for all tables referencing to the table you update or delete.

For example, suppose you have two tables, Class and Student. The Student table have a foreign key column, cid which references to the class_id column in table Class. When you want to update a class_id in Class table you will also need to update the cid column in Student table to maintain data integrity.

Suppose i want to change the class_id of Karate from 3 to 10. Since there is a row in Student table with cid value of 3, I have to update that row too.

```
$query = "UPDATE Class SET class_id = 10 WHERE class_id = 3";     mysql_query($query);

$query = "UPDATE Student SET cid = 10 WHERE cid = 3";                    mysql_query($query);
```

Below are the data in Table and Student class before an update query :

Table Class

class_id	**class_name**
1	Silat
2	Kungfu
3	Karate
4	Taekwondo

Table Student

student_id	**student_name**	**cid**
1	Uzumaki Naruto	1
2	Uchiha Sasuke	3
3	Haruno Sakura	2

Now the content of Table and Student class after the update query are:

Table Class

class_id	**class_name**
1	Silat
2	Kungfu
10	Karate
4	Taekwondo

Table Student

student_id	**student_name**	**cid**
1	Uzumaki Naruto	1
2	Uchiha Sasuke	10
3	Haruno Sakura	2

You can go as far as creating your own functions in PHP to ensure the data integrity. I have done this before and I hope you don't do it. Save yourself the headache and just write appropriate queries to maintain your data integrity whenever you update / delete rows from a table.

This means that whenever you make a query to update / delete **always** consult your database design to see if you need to update / delete another table to maintain data integrity. Your code will be more portable like this.

Using LOCK TABLES

When your web application is used by more than one user using LOCK TABLES before any update / delete query is a safe bet. This will make sure that only one user change the table at a time.

Using the above update code examples again, suppose there are two users. The first one want to update one row in Class table and the second want to delete it

```
$query = "LOCK TABLES Class WRITE, Student WRITE";      mysql_query($query);
$query = "DELETE FROM Class WHERE class_id = 3";      mysql_query($query);
$query = "DELETE FROM Student WHERE class_id = 3";   mysql_query($query);

$query = "UNLOCK TABLES";   mysql_query($query);
```

The update queries above can be rewritten as :

```
$query = "LOCK TABLES Class WRITE, Student WRITE";   mysql_query($query);

$query = "UPDATE Class SET class_id = 10 WHERE class_id = 3";   mysql_query($query);
$query = "UPDATE Student SET cid = 10 WHERE cid = 3";   mysql_query($query);

$query = "UNLOCK TABLES";     mysql_query($query);
```

By issuing the LOCK TABLES all other users are blocked from reading and writing to the tables. So you're update / delete query will continue to completion without any worries that the intended table already changed by another user

Uploading Files To MySQL Database

Using PHP to upload files into MySQL database sometimes needed by some web application. For instance for storing pdf documents or images to make some kind of online briefcase (like Yahoo briefcase).

For the first step, let's make the table for the upload files. The table will consist of.

1. id : Unique id for each file
2. name : File name
3. type : File content type
4. size : File size
5. content : The file itself

For column content we'll use BLOB data type. BLOB is a binary large object that can hold a variable amount of data. MySQL have four BLOB data types, they are :

TINYBLOB, BLOB, MEDIUMBLOB,LONGBLOB

Since BLOB is limited to store up to 64 kilobytes of data we will use MEDIUMBLOB so we can store larger files (up to 16 megabytes).

```
CREATE TABLE upload (
id INT NOT NULL AUTO_INCREMENT,
name VARCHAR(30) NOT NULL,
type VARCHAR(30) NOT NULL,
size INT NOT NULL,
content MEDIUMBLOB NOT NULL,
PRIMARY KEY(id)
);
```

Uploading a file to MySQL is a two step process. First you need to upload the file to the server then read the file and insert it to MySQL.

For uploading a file we need a form for the user to enter the file name or browse their computer and select a file. The input type="file" is used for that purpose.

```
<form method="post" enctype="multipart/form-data">
<table width="350" border="0" cellpadding="1" cellspacing="1" class="box">
<tr>
<td width="246">
<input type="hidden" name="MAX_FILE_SIZE" value="2000000">
<input name="userfile" type="file" id="userfile">
</td>
<td width="80"><input name="upload" type="submit" class="box" id="upload" value=" Upload "></td>
</tr>
</table>
</form>
```

An upload form **must** have encytype="multipart/form-data" otherwise it won't work at all. Of course the form method also need to be set to method="post". Also remember to put a hidden input MAX_FILE_SIZE **before** the file input. It's to restrict the size of files.

After the form is submitted the we need to read the autoglobal $_FILES. In the example above the input name for the file is userfile so the content of $_FILES are like this :

$_FILES['userfile']['name'] - The original name of the file on the client machine.

$_FILES['userfile']['type'] - The mime type of the file, if the browser provided this information. An example would be "image/gif".

$_FILES['userfile']['size'] - The size, in bytes, of the uploaded file.

$_FILES['userfile']['tmp_name'] - The temporary filename of the file in which the uploaded file was stored on the server.

$_FILES['userfile']['error'] - The error code associated with this file upload. ['error'] was added in PHP 4.2.0

```
<?php
if(isset($_POST['upload']) && $_FILES['userfile']['size'] > 0)
```

```
{
$fileName = $_FILES['userfile']['name'];
$tmpName  = $_FILES['userfile']['tmp_name'];
$fileSize = $_FILES['userfile']['size'];
$fileType = $_FILES['userfile']['type'];

$fp      = fopen($tmpName, 'r');
$content = fread($fp, filesize($tmpName));
$content = addslashes($content);
fclose($fp);

if(!get_magic_quotes_gpc())
{
   $fileName = addslashes($fileName);
}
$query = "INSERT INTO upload (name, size, type, content ) ".
"VALUES ('$fileName', '$fileSize', '$fileType', '$content')";

mysql_query($query) or die('Error, query failed');

echo "<br>File $fileName uploaded<br>";
}
?>
```

Before you do anything with the uploaded file. You **should not** assume that the file was uploaded successfully to the server. Always check to see if the file was successfully uploaded by looking at the file size. If it's larger than zero byte then we can assume that the file is uploaded successfully.

PHP saves the uploaded file with a temporary name and save the name in $_FILES['userfile']['tmp_name']. Our next job is to read the content of this file and insert the content to database. Always make sure that you use addslashes() to escape the content. Using addslashes() to the file name is also recommended because you never know what the file name would be.

That's it now you can upload your files to MySQL. Now it's time to write the script to download those files.

Downloading Files From MySQL Database

When we upload a file to database we also save the file type and length. These were not needed for uploading the files but is needed for downloading the files from the database.

The download page list the file names stored in database. The names are printed as a url. The url would look like download.php?id=3.

```
<html>
<head>
<title>Download File From MySQL</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body>
php?>
$query = "SELECT id, name FROM upload";
$result = mysql_query($query) or die('Error, query failed');
if(mysql_num_rows($result) == 0) echo "Database is empty <br>";

else {
while(list($id, $name) = mysql_fetch_array($result)) { ?>
<a href="download.php?id=<?php echo $id;?>"><?php echo $name;?></a> <br>
<?php
}
}
?>
</body>
</html>
```

When you click the download link, the $_GET['id'] will be set. We can use this id to identify which files to get from the database. Below is the code for downloading files from MySQL Database.

```
<?php   #Download.php
if(isset($_GET['id']))
{
// if id is set then get the file with the id from database

include 'library/config.php';
include 'library/opendb.php';

$id    = $_GET['id'];
$query = "SELECT name, type, size, content " .
         "FROM upload WHERE id = '$id'";

$result = mysql_query($query) or die('Error, query failed');
list($name, $type, $size, $content) = mysql_fetch_array($result);

header("Content-length: $size");
header("Content-type: $type");
header("Content-Disposition: attachment; filename=$name");
echo $content;
include 'library/closedb.php';
exit;
}

?>
```

Before sending the file content using echo first we need to set several headers. They are :

1. **header("Content-length: $size")**
   This header tells the browser how large the file is. Some browser need it to be able to download the file properly. Anyway it's a good manner telling how big the file is. That way anyone who download the file can predict how long the download will take.

2. **header("Content-type: $type")**
   This header tells the browser what kind of file it tries to download.

3. **header("Content-Disposition: attachment; filename=$name");**
   Tells the browser to save this downloaded file under the specified name. If you don't send this header the browser will try to save the file using the script's name (download.php).

After sending the file the script stops executing by calling exit.

NOTE :
When sending headers the most common error message you will see is something like this :

Warning: Cannot modify header information - headers already sent by (output started at C:\xampp\htdocs\config.php:7) in C:\xampp\htdocs\download.php on line 13

This error happens because some data was already sent before we send the header. As for the error message above it happens because i "accidentally" add one space right after the PHP closing tag (?>) in config.php(The file included in the file before a header statement.) file. So if you see this error message when you're sending a header just make sure you don't have any data sent before calling header(). Check the file mentioned in the error message and go to the line number specified.