PHP/MySQL

Michael Powell

Basic PHP

Echo Statement:

```
<?php echo “Hello World”; ?>
```

Variable and Echo:

```
<?php $var=“Hello World”; echo $var; ?>
```

Include:

```
<?php include(menu.php); ?>
```

MySQL: Things You Need

- Database
  - User Name
    - Do not use “root”
  - Password

- Tables in the Database
  - Records/Rows
  - Fields/Columns

Databases

- 1 Database can be assigned to a whole site.
- If sections of a website do not need to interact, they can run just fine with separate databases.
- Separate databases/tables for each user is better for security.

Tables

- Normally Required:
  - Table for Users – If the site has tracking or is interactive you will need a users table.
  - Table for IPs/Logging – Can keep you from having to look at server logs if security becomes an issue.
- Optional:
  - Submissions – Comments or content can be stored.
  - Web Pages – Pages can be stored and called on when a user accesses a certain link.

Columns

- Think of everything you might need in the future, not what is just required now. This can save you time from having to change coding later.
- Do not overlap fields. I will not go into depth about this, but you should not have multiple fields in different tables storing the same information. You can call the information through a unique identifier.

PHP

- When writing the pages decide what will be duplicated on multiple pages, and put that information into a file that can be included.
- If you believe the information is going to change often, then pulling it from a database is normally a better solution.
- Do not use PHP, if it isn't needed. If you are doing a simple web page keep it HTML.

HTML Forms

- Text input is the most common.
- Decide if you want a list or set of options for the visitors, or if you trust them with what they are putting in.
- Make sure items that are required are marked required, and return an appropriate error if not completed.

HTML Form Example

Basic Form Example:

```
<form METHOD="POST" action="process">
<input type="text">
<input type="submit" value="Submit">
</form>
```

“process” would actually be process.php. This page would handle the input from this form.

MySQL Database Connection

- The following commands show you how to connect, and to generate an error if the connection fails.
- They also show how to select a database.
- Once you are done you can then “close” the connection to the database.

MySQL Connect

```
$connection =
   mysql_connect("localhost","user","password");

if (!$connection)
  {
  die('Could not connect: ' . mysql_error());
  }
```

MySQL Database Selection

- You need to select a database before you can gather data from it or modify the data it has.

```
mysql_select_db("database", $connection);
```

MySQL Close Connection

- This command is not necessary in all situations, but if you want to close a connection this command will do that.

```
mysql_close($connection);
```

MySQL Insertion

- First, make sure anything being inserted has been “escaped” so no commands are issued to the database.
- Second, make sure you put limitations on the database about what can be stored so invalid information can not be inserted.

MySQL Escape

- The escape string adds a leading \ to any symbols used in MySQL commands, this prevents malicious code from being executed through HTML forms.

Example:

```
$var = mysql_real_escape_string($var);
```

MySQL Insert

- You can enter similar commands to what PHPMyAdmin generates into mysql_query() to get the same results.
- When inserting variables make sure you have the same number of variables as there are columns in the database.

Example:

```
mysql_query("INSERT INTO table
   VALUES('$var1', '$var2',  '$var3')");
```

MySQL Insert

- If you have problems with your Insert command, echo what is being executed. You can copy and paste the command into PHPMyAdmin and it will give you an explanation of the possible problem.
- If PHPMyAdmin does not return an error, then it is probably another part of your code.

MySQL Delete

- The delete command can potentially wipe everything from your table if it is not used properly. Make sure your WHERE statement is correct!

```
mysql_query("DELETE FROM `table` WHERE
  `table`.`VAR1` = '$var1'");
```

MySQL Update

- If you are changing a lot of variables, or if the number of variables that can change is dynamic, I normally delete the entry and add it back with all the new information.

Example:

```
mysql_query("UPDATE table SET Column1 =
    '$var' WHERE Column2 = '$var2'");
```

MySQL Select

- After you have a connection to the database, you can select what information you want from the database with the following command.
- The following command stores the select statement results to the $result variable.

```
$result = mysql_query("SELECT * FROM table");
```

MySQL Select

- You can cycle through all returned entries using a while loop.
- The below code displays the result from 1 column, then makes a line break.

```
while($row = mysql_fetch_array($result))
  {
  echo $row['Column2'];
  echo “<br>”;
  }
```

Checks

- Often it is good to have a check in place to make sure if nothing is being returned, the user knows that and doesn't assume there was an error.
- The below code echoes a message if nothing is returned.

```
if (mysql_num_rows($result) != 0) {
//Action code
}
else echo “Sorry no results were returned.”;
```

References/Resources

- http://w3schools.com/php/default.asp
- http://www.php.net/manual/en/