webmonkey/programming/

PHP/MySQL Tutorial

by Graeme Merrall

In this lesson, we're going to dive right in and create some simple yet useful pages using PHP and MySQL. Let's start by displaying the database we created yesterday, but with a little more panache.

First, let's query our database using the following code.

```
<html>

<body>

<?php



$db = mysql_connect("localhost", "root");

mysql_select_db("mydb",$db);

$result = mysql_query("SELECT * FROM employees",$db);

echo "<table border=1>\n";

echo "<tr><td>Name</td><td>Position</tr>\n";

while ($myrow = mysql_fetch_row($result)) {

        printf("<tr><td>%s %s</td><td>%s</td></tr>\n", $myrow[1], $myrow[2], $myrow[3]);



}

echo "</table>\n";



?>
```

```
</body>


</html>
```

You probably noticed that we introduced a couple of new features here. Most obvious is the `while()` loop. The loop says that as long as there are new rows of data to be grabbed (using the `mysql_fetch_row()` function), assign that row to the `$myrow` variable. Then execute the instructions between the curly brackets ({}). Take a look for a second, and this should make sense.

The `mysql_fetch_row()` function bears a closer look. One small problem with `mysql_fetch_row()` is that it returns an array that supports only numeric references to the individual fields. So the first field is referred to as 0, the second as 1, and so on. With complex queries this can become something of a nightmare.

Now let's examine the loop in more detail. The first few lines you'll recognize from the example in Lesson 1. Then in the `while()` loop we fetch a row from the result and assign it to the array `$myrow`. Then we print the contents of the array on the screen with the `printf` function. After that it loops around again, and another row is assigned to `$myrow`. It will do this until it runs out of rows to grab.

The great thing about a `while()` loop is that if your query returns no records, you won't get an error message. The first time through there won't be any data to assign to `$myrow`, and the program will just move on.

But if the query returns no data, we have no way of letting the user know, and we should probably provide some sort of message. This is possible, so let's do it.

Stay Informed

Take a look at this script.

```
<html>


<body>


<?php


$db = mysql_connect("localhost", "root");

mysql_select_db("mydb",$db);

$result = mysql_query("SELECT * FROM employees",$db);

if ($myrow = mysql_fetch_array($result)) {

  echo "<table border=1>\n";

  echo "<tr><td>Name</td><td>Position</td></tr>\n";

  do {
```

```
    printf("<tr><td>%s %s</td><td>%s</td></tr>\n", $myrow["first"], $myrow["last"], $myrow["address"]);

  } while ($myrow = mysql_fetch_array($result));

          echo "</table>\n";

} else {

          echo "Sorry, no records were found!";

}


?>


</body>


</html>
```

There are a number of new features introduced here, but they're quite simple. First, there's the `mysql_fetch_array()` function. This is exactly the same as `mysql_fetch_row()` with one nice exception: Using this function, we can refer to fields by their names (such as `$myrow["first"]`) rather than their numbers. This should save us some headaches. We've also introduced a do/while loop and an if-else statement.

The if-else statement says that if we can assign a row to `$myrow`, then continue; otherwise skip to the `else` section and do what's in there.

The do/while loop is a variation of the `while()` loop we used on the last page. We need the do/while loop here for a very good reason: With the initial if statement, we assigned the first row returned by the query to the variable `$myrow`. If at this point we executed a regular while statement (such as `while ($myrow = mysql_fetch_row($result)`), we'd be kicking the first record out of the variable and replacing it with the second record. But the do/while loop lets us test the condition after the code has been run once. So there's no chance of us accidentally skipping a row.

Finally, if there are no records returned at all, the statements contained in the `else{}` portion will be executed. To see this portion in action, change the SQL statement to `SELECT * FROM employees WHERE id=6` or something else that will return no records.

Now let's extend this looping and if-else code to make one fancy-schmancy page. You're going to love it.

Link Intelligently

We're going to take that looping power we just learned and use it in a more practical example. But before we proceed here, you should know how to work with forms, the querystring, and the GET and POST methods. Jay covered this not long ago, so go take a look at his article if this is unfamiliar to you.

Right now I'm going to work with the querystring. As you should know, there are three ways to get information into the querystring. The first is to use the GET method in a form. The second is to type the information into the URL on your browser. And third, you can embed a querystring in a standard link. Just make the anchor tag look something like this: <a href="http://my_machine/mypage.php3?id=1">. We're going to use this technique right now.

First off, lets query our database again and list the employee names. Look at the following script. Much of this should look pretty familiar by now.

```
<html>

<body>

<?php


$db = mysql_connect("localhost", "root");

mysql_select_db("mydb",$db);

$result = mysql_query("SELECT * FROM employees",$db);

if ($myrow = mysql_fetch_array($result)) {

  do {

    printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF, $myrow["id"], $myrow["first"], $myrow["

  } while ($myrow = mysql_fetch_array($result));

} else {

  echo "Sorry, no records were found!";

}

?>


</body>


</html>
```

Everything's about the same except the `printf` function, so let's look at it in some detail.

First notice that each quotation mark is preceeded by a backslash. The backslash tells PHP to display the character following it, rather than treat it as part of the code. Also note the use of the variable $PHP_SELF. This variable, which stores the script's name and location, is passed along with every PHP page. It's helpful here because we just want this file to call itself. Using $PHP_SELF, we can be sure that will happen, even if the file is moved to another directory - or even another machine.

As I just mentioned, these links will recall the page. On the second time through, however, some information will be added

to the querystring.

PHP does a nifty thing when it sees a name=value pair in the querystring. It automatically creates a variable with the name and value the querystring indicated. This feature allows us to test if it's the first or second time through this page. All we have to do is ask PHP if the variable `$id` exists.

Once I know the answer to that question, I can display a different set of information the second time through. Here's how:

```
<html>
<body>
<?php

$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
// display individual record
if ($id) {
   $result = mysql_query("SELECT * FROM employees WHERE id=$id",$db);
   $myrow = mysql_fetch_array($result);
   printf("First name: %s\n<br>", $myrow["first"]);
   printf("Last name: %s\n<br>", $myrow["last"]);
   printf("Address: %s\n<br>", $myrow["address"]);
   printf("Position: %s\n<br>", $myrow["position"]);
} else {
    // show employee list
   $result = mysql_query("SELECT * FROM employees",$db);
    if ($myrow = mysql_fetch_array($result)) {
      // display list if there are records to display
      do {
        printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF, $myrow["id"], $myrow["first"], $myr
      } while ($myrow = mysql_fetch_array($result));
    } else {
      // no records to display
      echo "Sorry, no records were found!";
    }
}

?>

</body>
```

```
</html>
```

This code is getting complex now, so I've started to use comments to keep track of what's going on. You can use // to make a single-line comment or /* and */ to start and end a large comment block.

And there we have it: your first truly useful PHP/MySQL script! Now let's take a look at how to plug forms into it and send information back into the database.

Throw in Some Forms

We've managed to get data from a database without much difficulty. But what about sending data the other way? It's not a problem with PHP.

First let's create a page with a simple form.

```
<html>
<body>

<form method="post" action="<?php echo $PHP_SELF?>">
First name:<input type="Text" name="first"><br>
Last name:<input type="Text" name="last"><br>
Address:<input type="Text" name="address"><br>
Position:<input type="Text" name="position"><br>
<input type="Submit" name="submit" value="Enter information">
</form>

</body>
</html>
```

Note the use of $PHP_SELF again. Like I said in Lesson 1, you can use PHP anywhere inside your HTML code. You'll also notice that each form element matches the field name in the database. This is not compulsory; it's just a good idea so you can get your head around your code later on.

Also notice that I've given the Submit button a `name` attribute. I've done this so I can test for the existence of a `$submit` variable. That way, when the page is called again, I'll know whether someone used this form.

I should mention that you don't have to have a page that loops back on itself. You can span two, three, or more pages, if you like. This way everything stays compact.

OK, let's add some code that will check for the form input. Just to prove that the form input does make it through, I'll dump all

the variables to the screen with `$HTTP_POST_VARS`. This is a useful debugging feature. If you ever need to see all the variables on a page, use `$GLOBALS`.

```
<html>
<body>
<?php

if ($submit) {
  // process form
  while (list($name, $value) = each($HTTP_POST_VARS)) {
    echo "$name = $value<br>\n";
  }
} else{
  // display form
  ?>
  <form method="post" action="<?php echo $PHP_SELF?>">
  First name:<input type="Text" name="first"><br>
  Last name:<input type="Text" name="last"><br>
  Address:<input type="Text" name="address"><br>
  Position:<input type="Text" name="position"><br>
  <input type="Submit" name="submit" value="Enter information">
  </form>
  <?php

} // end if

?>

</body>

</html>
```

Now that this is looking good, let's take the form information and post it to the database.

```
<html>
<body>

<?php

if ($submit) {
  // process form
```

```
  $db = mysql_connect("localhost", "root");
  mysql_select_db("mydb",$db);
  $sql = "INSERT INTO employees (first,last,address,position) VALUES ('$first','$last','$address','$p
  $result = mysql_query($sql);
  echo "Thank you! Information entered.\n";
} else{

  // display form

  ?>

  <form method="post" action="<?php echo $PHP_SELF?>">
  First name:<input type="Text" name="first"><br>
  Last name:<input type="Text" name="last"><br>
  Address:<input type="Text" name="address"><br>
  Position:<input type="Text" name="position"><br>
  <input type="Submit" name="submit" value="Enter information">
  </form>

  <?php

} // end if

?>

</body>

</html>
```

You've now inserted data into the database. It's still far from perfect. What if someone leaves a field blank or enters text when we want a numeric entry? What if there's an error somewhere?

Don't worry. We'll get to that.

Make the Forms Smarter

Throughout this tutorial, I've been loading the SQL statement into a variable (`$sql`) before firing the query at the database with `mysql_query()`. This is useful for debugging. If something goes wrong, you can always echo the SQL to the screen to examine it for mistakes.

We already know how to get data into the database. Now let's try modifying records that are already in the database. Editing data combines two elements we've already seen: displaying data on the screen and sending data back to the database via

form input. However, editing is slightly different in that we have to show the appropriate data in the form.

First, let's recycle the code from Lesson 1 to display the employee names on our page. But this time through, we're going to populate our form with employee information. It should look a little like this:

```
<html>
<body>
<?php

$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
if ($id) {
  // query the DB
  $sql = "SELECT * FROM employees WHERE id=$id";
  $result = mysql_query($sql);
  $myrow = mysql_fetch_array($result);
  ?>

  <form method="post" action="<?php echo $PHP_SELF?>">
  <input type=hidden name="id" value="<?php echo $myrow["id"] ?>">
  First name:<input type="Text" name="first" value="<?php echo $myrow["first"] ?>"><br>
  Last name:<input type="Text" name="last" value="<?php echo $myrow["last"] ?>"><br>
  Address:<input type="Text" name="address" value="<?php echo $myrow["address"] ?>"><br>
  Position:<input type="Text" name="position" value="<?php echo $myrow["position"] ?>"><br>
  <input type="Submit" name="submit" value="Enter information">
  </form>

  <?php

} else {
  // display list of employees
  $result = mysql_query("SELECT * FROM employees",$db);
  while ($myrow = mysql_fetch_array($result)) {
    printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF, $myrow["id"], $myrow["first"], $myrow["
  }
}

?>

</body>
</html>
```

We just echoed the field information into the `value` attribute of the each element, which was fairly easy. Let's build on this a little more. We will add the ability to send the edited code back to the database. Again, we're going to use the Submit button to test whether we need to process the form input. Also note the slightly different SQL statement we use.

```
<html>
<body>
<?php

$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);
if ($id) {
  if ($submit) {
    $sql = "UPDATE employees SET first='$first',last='$last',address='$address',position='$position'
    $result = mysql_query($sql);
    echo "Thank you! Information updated.\n";
  } else {
    // query the DB
    $sql = "SELECT * FROM employees WHERE id=$id";
    $result = mysql_query($sql);
    $myrow = mysql_fetch_array($result);
    ?>

    <form method="post" action="<?php echo $PHP_SELF?>">
    <input type=hidden name="id" value="<?php echo $myrow["id"] ?>">
    First name:<input type="Text" name="first" value="<?php echo $myrow["first"] ?>"><br>
    Last name:<input type="Text" name="last" value="<?php echo $myrow["last"] ?>"><br>
    Address:<input type="Text" name="address" value="<?php echo $myrow["address"] ?>"><br>
    Position:<input type="Text" name="position" value="<?php echo $myrow["position"] ?>"><br>
    <input type="Submit" name="submit" value="Enter information">
    </form>
          <?php
          }
} else {

  // display list of employees
  $result = mysql_query("SELECT * FROM employees",$db);
  while ($myrow = mysql_fetch_array($result)) {
    printf("<a href=\"%s?id=%s\">%s %s</a><br>\n", $PHP_SELF, $myrow["id"], $myrow["first"], $myrow["
  }
}
```

```
?>


</body>


</html>
```

And that's that. We've managed to combine most of the features we've seen into one script. You can also see how we've used an `if()` statement inside another `if()` statement to check for multiple conditions.

It's time to put it all together and make one killer script.

All Together Now

We'll finish up this lesson by putting everything into a single page that can add, edit, and remove entries from the database. It's an extension of what we've covered so far and makes for a good review. Let's take a look.

```
<html>
<body>


<?php


$db = mysql_connect("localhost", "root");
mysql_select_db("mydb",$db);


if ($submit) {
  // here if no ID then adding else we're editing
  if ($id) {
    $sql = "UPDATE employees SET first='$first',last='$last',address='$address',position='$position'
  } else {
    $sql = "INSERT INTO employees (first,last,address,position) VALUES ('$first','$last','$address','
  }
  // run SQL against the DB
  $result = mysql_query($sql);
  echo "Record updated/edited!<p>";
} elseif ($delete) {
          // delete a record
    $sql = "DELETE FROM employees WHERE id=$id";
    $result = mysql_query($sql);
    echo "$sql Record deleted!<p>";
} else {
  // this part happens if we don't press submit
  if (!$id) {
```

```
  // print the list if there is not editing
  $result = mysql_query("SELECT * FROM employees",$db);
  while ($myrow = mysql_fetch_array($result)) {
    printf("<a href=\"%s?id=%s\">%s %s</a> \n", $PHP_SELF, $myrow["id"], $myrow["first"], $myrow["l
          printf("<a href=\"%s?id=%s&delete=yes\">(DELETE)</a><br>", $PHP_SELF, $myrow["id"]);
  }
}

?>
<P>
<a href="<?php echo $PHP_SELF?>">ADD A RECORD</a>
<P>
<form method="post" action="<?php echo $PHP_SELF?>">
<?php

if ($id) {
  // editing so select a record
  $sql = "SELECT * FROM employees WHERE id=$id";
  $result = mysql_query($sql);
  $myrow = mysql_fetch_array($result);
  $id = $myrow["id"];
  $first = $myrow["first"];
  $last = $myrow["last"];
  $address = $myrow["address"];
  $position = $myrow["position"];
  // print the id for editing

  ?>
  <input type=hidden name="id" value="<?php echo $id ?>">
  <?php
}

?>
First name:<input type="Text" name="first" value="<?php echo $first ?>"><br>
Last name:<input type="Text" name="last" value="<?php echo $last ?>"><br>
Address:<input type="Text" name="address" value="<?php echo $address ?>"><br>
Position:<input type="Text" name="position" value="<?php echo $position ?>"><br>
<input type="Submit" name="submit" value="Enter information">
</form>

<?php

}
```

```
?>


</body>

</html>
```

This looks complex, but it really isn't. The script is broken up into three parts. The first `if()` statement checks to see whether the Submit button has been pressed, and if it has, it checks to see whether the variable `$id` exists. If doesn't, then we're adding a record. Otherwise, we're editing a record.

Next we check to see whether the variable `$delete` exists. If it does, we delete a record. Note that with the first `if()` statement we checked for a variable that came through as a POST, and in this one, the variable would be part of a GET.

Finally, we take the default action that displays the list of employees and the form. Again we check for the existence of the `$id` variable. If it exists, we query the database to display the relevant record. Otherwise, we display a blank form.

We've now put all we've learned into one script. We used `while()` loops and `if()` statements, and we ran the gamut of the basic SQL statements - SELECT, INSERT, UPDATE, and DELETE. Lastly, we've looked at how we can pass information from one page to another using URLs and form input.

In Lesson 3 we'll look at how to make the page more intelligent.

Graeme Merrall works in New Zealand for KC Multimedia, an Internet/intranet design company. While sometimes being forced to use ASP, he enjoys spending his time coding in PHP, keeping the grass trimmed, and scaring his work mates with song lyrics repeated out of context.