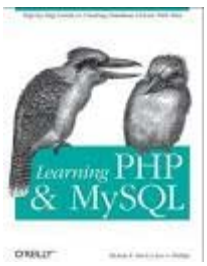
MYSQL RSS

Working with PHP and MySQL

By: O'Reilly Media
Rating: ★★★★☆ / 14
2007-05-24

Last week, you began learning how to use PHP to display and modify data from a MySQL database. This week, you'll learn how to select the database, fetch and display data, and more. This article is excerpted from chapter 9 of *Learning PHP and MySQL*, written by Michele Davis and Jon Phillips (O'Reilly, 2006; ISBN: 0596101104). Available from booksellers or direct from O'Reilly Media.



Selecting the Database

Now that you're connected, the next step is to select which database to use with the `mysql_select_db` command. It takes two parameters: the database name and, optionally, the database connection. If you don't specify the database connection, the default is the connection from the last `mysql_connect`.

```
$db_select = mysql_select_db($db_database);
if (!$db_select){
```

```
    die ("Could not select the database: <br />". mysql_error());
  }
```

Again, it's good practice to check for an error and display it every time you access the database.

> While it's possible to call `mysql_select_db` multiple times within the same script, it's not considered good practice. Generally, you should be able to do all of your work with one database. Maintaining connections and results to multiple databases are beyond this book's scope.

Now that you've got a good database connection, you're ready to execute your SQL query.

Building the SQL SELECT Query

Building a SQL query is as easy as setting a variable to the string that is your SQL query. Of course, you'll need to use a valid SQL query, or MySQL returns with an error when you execute the query. The variable name `$query` is used, but you can choose anything you'd like for a variable name. The SQL query in this example is `SELECT * FROM` books .

> Unlike when you used the `mysql` command-line client, the query does not have a semicolon at the end.

You can build up your query in parts using the string concatenate (`.`) operator:

```
  $select = ' SELECT ';
  $column = ' * ';
  $from = ' FROM ';
  $tables = ' `books` ';
  $where = '';
  $query = $select.$column.$from.$tables.$where;
```

Which is a more flexible version of this:

```
  $query = "SELECT * FROM books";
```

The query string could also use a variable in the `WHERE` clause to limit which rows are returned based on user information or another query.

Now that you have your query assigned to a variable, you can execute it.

Executing the Query

To have the database execute the query, use the `mysql_query` function. It takes two parameters--the query and optionally the database link--and returns the result. Save a link to the results in a variable called, you guessed it, `$result`! This is also a good place to check the return code from `mysql_query` to make sure that there were no errors in the query string or the database connection by verifying that `$result` is not `FALSE`.

```
  $result = mysql_query( $query );
  if (!$result)
  {
  die ("Could not query the database:
<br />". mysql_error());
  }
```

When the database executes the query, all of the results form a result set. These correspond to the rows that you saw upon doing a query using the `mysql` command-line client. To display them, you process each row, one at a time.

Use `mysql_fetch_row` to get the rows from the result set. It takes the result you stored in `$result` from the query as a parameter. It returns one row at a time from the query until there are no more rows, and then it returns `FALSE`. Therefore, you do a loop on the result of `mysql_fetch_row` and define some code to display each row:

```
  while ($result_row = mysql_fetch_row($result)){
        echo $result_row[2] . '
<br />';
  }
```

Fetch types

This is not the only way to fetch the results. Using `mysql_fetch_array`, PHP can place the results into an array in one step. It

takes a result as its first parameter, and the way to bind the results as an optional second parameter. If MYSQL_ASSOC is specified, the results are indexed in an array based on their column names in the query. If MYSQL_NUM is specified, then the number starting at zero accesses the results. The default value, MYSQL_BOTH, returns a result array with both types. The mysql_fetch_assoc is an alternative to supplying the MYSQL_ASSOC argument.

If you rewrote the code above to use mysql_fetch_array with an associative indexed array, it would look like this:

```
while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {
                                    echo $row[title]. '<br />';
}
```

Closing the Connection

As a rule of thumb, you always want to close a connection to a database when you're done using it. Closing a database with mysql_close will tell PHP and MySQL that you no longer will be using the connection, and will free any resources and memory allocated to it.

```
mysql_close($connection)
```

Now you're going to take all of the steps and put them into a single PHP file that you'll call *db_test.php*. You should place the PHP script shown in Example 9-5 in the same directory as the *db_login.php* file.

Example 9-5. *Displaying the books and authors*

```
<?php
// Include our login information
include('db_login.php');
// Connect
$connection = mysql_connect( $db_host, $db_username, $db_password );
if (!$connection)
{
   die ("Could not connect to the database:
<br />". mysql_error());
}
// Select the database $db_select=mysql_select_db($db_database); if (!$db_select)
{
   die ("Could not select the database:
<br />". mysql_error());
}
// Assign the query
$query = "SELECT * FROM `books` NATURAL JOIN `authors`";
// Execute the query
$result = mysql_query( $query );
if (!$result)
{
   die ("Could not query the database:
<br />". mysql_error());
}

// Fetch and display the results
while ($result_row = mysql_fetch_row(($result)))
{

       echo 'Title: '.$result_row[1] . '
<br />';
       echo 'Author: '.$result_row[4] . '
<br />';
       echo 'Pages: '.$result_row[2] . '
<br />';
}
/ /Close the connection
mysql_close($connection);
?>
```

Here's the output from Example 9-5:

```
  Title: Linux in a Nutshell<br />Author: Ellen Siever<br /> Pages: 476<br />
  <br />Title: Linux in a Nutshell<br />
Author: Aaron Weber<br /> Pages: 476<br />
  <br />Title: Classic Shell Scripting<br> Author: Arnold Robbins<br /> Pages: 256<br />
  <br />Title: Classic ShellScripting<br />
Author: Nelson H.F. Beebe<br /> Pages:
256<br /><br />
```

This displays in your browser as in Figure 9-3.

If you don't see the screen in Figure 9-3, then you'll see an error from whichever step in the process had a problem, giving you an idea of what went wrong and where it was wrong.

To make the display more appealing, you can put the information into a table, as shown in Example 9-6. You also add complete HTML headers.

Figure 9-3. *How Example 9-5 displays in the browser*

Example 9-6. *Displaying the results of a query in an HTML table*

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html401
/loose.dtd"> <html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"> <title>Displaying in ar
HTML table</title> </head>
<body>
<table border="1">
<tr>
<th>Title</th>
<th>Author</th>
<th>Pages</th>
</tr>
<?php
//Include our login information
include('db_login.php');
// Connect
$connection = mysql_connect($db_host, $db_username, $db_password);
if (!$connection){
die("Could not connect to the database:
<br />". mysql_error());
}
// Select the database
$db_select = mysql_select_db($db_database); if (!$db_select){
die ("Could not select the database:
```

```
<br />". mysql_error());
}
// Assign the query
$query = "SELECT * FROM `books` NATURAL JOIN `authors`";
// Execute the query
$result = mysql_query($query);
if (!$result){
die ("Could not query the database:
<br />". mysql_error());
}
// Fetch and display the results
while ($row = mysql_fetch_array($result, MYSQL_ASSOC)){
$title = $row["title"];
$author = $row["author"];
$pages = $row["pages"];
echo "<tr>";
echo "<td>$title</td>";
echo "<td>$author</td>";
echo "<td>$pages</td>";
echo "</tr>";
}
// Close the connection
mysql_close($connection);
?>
</table>
</body>
</html>
```

Example 9-6. displays in your browser as shown in Figure 9-4.

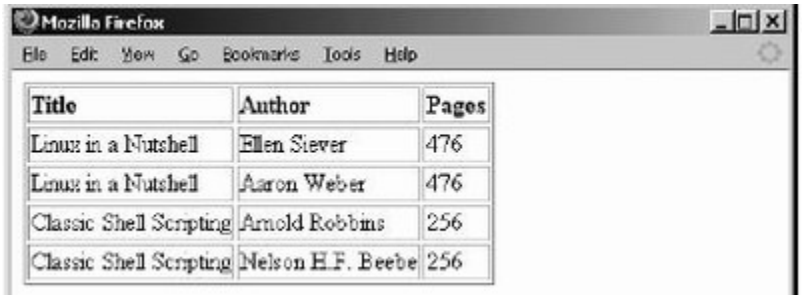


Figure 9-4. *The same data but in an HTML table*

Notice that you made use of the `MYSQL_ASSOC` fetch type in Example 9-6. You're probably saying to yourself, "That's great, but how do I display the book titles with the authors all on one line?" This is where we talk about PEAR.

PEAR is a framework and distribution system for reusable PHP components. Actually, PEAR is a collection of add-on functionality for PHP development. There are many modules available to handle everything from session management to shopping cart functionality. Modules that are currently available are listed in Table 9-1.

Table 9-1. *Pear modules*

Authentication	HTML	Processing
Benchmarking	HTTP	Science
Caching	Images	Semantic Web
Configuration	Internationalization	Streams
Console	Logging	Structures
Database	Mail	System

Date/Time	Math	Test
Encryption	Networking	Tools & Utilities
Event	Numbers	Validate
File Formats	Payment	Web Services
File System	PEAR	XML
GTK Components	PHP	

Our list is not complete. Visit *http://pear.php.net* to find out all of the modules that are available for download.

Installing

PEAR uses a Package Manager to manage which PEAR features you install. Whether you need to install the Package Manager depends on which version of PHP you installed. If you're running PHP 4.3.0 or newer, it's already installed. If you're running PHP 5.0, PEAR has been split out into a separate package. The DB package that you're interested in is also installed by default with the Package Manager. So if you have the Package Manger, you're all set.

Unix

You can install the Package Manager on a Unix system by executing the following from the shell (command-line) prompt:

```
lynx -source http://go-pear.org/ | php
```

This takes the output of the *go-pear.org* site (which is actually the source PHP code) to install PEAR and passes it along to the `php` command for execution.

Windows

The PHP 5 installation includes the PEAR installation script as *C:\php\go-pear.bat*. In case you didn't install all the files in Chapter 2, go ahead and extract all the PHP files to *C:/php* from the command prompt, and execute the *.bat* file. Figure 9-5 shows the initial screen after executing the PEAR installer.

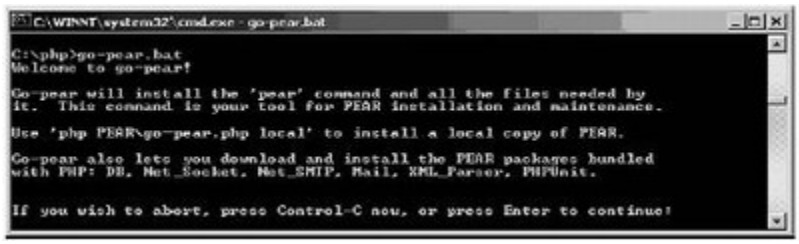


Figure 9-5. *The go-pear.bat install script*

You'll be asked a set of questions about paths. You can accept the defaults for all of them.

> The *php.exe* file must be in your path. Verify by typing `php.exe` from a command prompt. If it is not found, you'll need to add it to your `PATH` variable. To access your system path, navigate to Start -> Control Panel -> System -> Environment and add an entry to the end of the path with `C:\php`.

The PEAR installer creates a file called *C:\php\PEAR_ENV.reg.* You need to double-click to set up the PEAR paths in the registry. This file is contingent on which PEAR version you installed. When the dialog box appears to verify your information, you will add this to the registry and click OK.

You may have to edit the *php.ini* file after running this *.bat* file to add the PEAR directory to the include path. Line 447 of *php.ini* now looks like this:

```
include_path = ".;c:\php\includes;c:\php\PEAR'
```

Apache must be restarted before the DB package can be used.

Please check back next week for the conclusion to this article.