O'REILLY
ONLamp.com
LAMP: THE OPEN SOURCE WEB PLATFORM

Published on **ONLamp.com** (http://www.onlamp.com/)
See this if you're having trouble printing code examples

Using MySQL from PHP, Part 2

by John Coggeshall
03/18/2004

Hello again, welcome back to PHP Foundations. In my last column, I introduced some of the basic functions used when accessing a MySQL database from PHP. With what you have learned so far, you should now be able to perform queries and access the result sets from within your PHP scripts. However, there are still several things to learn. Today I'll explain dealing with errors, determining the number of rows in a result set, and more. Let's get started.

Useful MySQL/PHP Functions

Dealing with Errors When Working with MySQL

Now that you know how to connect to, query, and work with databases, it's time to discuss what to do when things go wrong, as they surely will. Any time an error occurs when PHP is interacting with the database (except when you are attempting to establish a connection to the database), you can find the last error by using the `mysql_error()` and `mysql_errno()` functions:

```
mysql_error([$link]);
mysql_errno([$link]);
```

where the optional parameter `$link` represents the database connection for which to retrieve the last error message. These two functions will both report different details regarding the last error. `mysql_error()` provides a string error message and `mysql_errno()` returns the actual integer error code.

To illustrate the use of these functions, consider the following small code snippet. Assume `$link` is a valid database connection:

```
$result = mysql_query("SELECT * FROM foo", $link);
if (!$result) {
        $errno = mysql_errno($link);
        $error = mysql_error($link);

        echo "Database Error ($errno): $error";
}
```

Assuming that the table `foo` does not exist in the current database, the following output will display to the browser:

```
Database Error (1146):
Table 'fundamentals.foo' does not exist
```

Determining Result Counts

In my previous column, I introduced to you the family of functions that retrieve the rows of a result set into various formats within PHP. There are two particularly important pieces of information for which you lack a reasonable way to retrieve them: the number of rows in the result set, and the number of rows affected by a query. Fortunately, there are functions for both.

The `mysql_num_rows()` function returns the total number of rows within the result set for a particular query. Its syntax is as follows:

```
mysql_num_rows($result);
```

where the `$result` parameter is the result resource returned by a successful `mysql_query()` call. This function has several uses. Perhaps it's most useful to determine if a properly formed query actually returned any data at all. Create and execute a query, then test to see if `mysql_num_rows()` returns a number greater than zero.

While `mysql_num_rows()` is useful for `SELECT` statements, all SQL queries (especially `UPDATE`, `INSERT`, and `DELETE`) actually return a result. That's where the `mysql_affected_rows()` function comes in handy. This function retrieves the number of rows affected by the last query. It has the following syntax:

```
mysql_affected_rows([$link]);
```

where the optional parameter `$link` is the resource representing the database connection. Notice that, unlike the `mysql_num_rows()` function, which accepts a result resource as the parameter (meaning it can be calculated at any time), the `mysql_affected_rows()` function must be called immediately following the query for which you need the information.

Here are examples of both functions and their use:

```
<?php

/* Assume $link is a valid database connection
  and a database has been selected */

$result = mysql_query("SELECT * FROM books", $link);

if (!$result) die("Query Failed.");

if (mysql_num_rows($result) > 0) {
```

```
        mysql_query("UPDATE books SET author_id=2 WHERE book_id=1", $link);
        $affected = mysql_affected_rows($link);

         echo "Number of records modified: $affected";
}
?>
```

Other Useful PHP/MySQL Functions

You'll be happy to know that we are approaching the end of function introductions. Soon, I'll put all of these functions to good use developing a database-driven application. However, before we can get to that, I'd like to introduce two more functions. These functions are not related to each other, but for lack of a better place to put them, I've decided to introduce them both to you at once.

The `mysql_insert_id()` function retrieves the last ID used when inserting a row into a table with an auto-increment column. Its syntax is identical to that of `mysql_affected_rows()`:

```
mysql_insert_id([$link]);
```

where the optional parameter `$link` is the resource representing the database connection. As was the case with the `mysql_affected_rows()` function, you must call `mysql_insert_id()` immediately following the call to the `mysql_query()` function that executed the `INSERT` statement in question.

Note: This function duplicates functionality found in the `LAST_INSERT_ID()` SQL function, and due to certain limitations, may produce unpredictable results if it operates on a `BIGINT` column. If you need the functionality of `mysql_insert_id()` in these circumstances, use the SQL function instead.

The other function is `mysql_escape_string()`. This function is extremely convenient and important when working with databases. It works similarly to the `addslashes()` PHP function that escapes special characters in a string, except that it works specifically with the MySQL database. The syntax is as follows:

```
mysql_escape_string($string)
```

where `$string` is the string to escape. This function plays an important role in designing database-driven applications that execute queries based on input from the user. Because you cannot trust user input not to do something nasty (whether deliberately or accidentally), use this function to escape any special characters whenever you insert strings into the database.

More PHP/MySQL to Come

That's all for today. With today's column, you should have everything you need to begin developing database-driven web applications! My next column will explain the development of a front end to the book and author information database I used during the crash-course on MySQL I discussed earlier in the series, complete with adding, removing, and digging into the data within the database -- all from a web interface! See you then.

John Coggeshall is a a PHP consultant and author who started losing sleep over PHP around five years ago.

---

*Read more **PHP Foundations** columns.*

Return to the PHP DevCenter.