Accessing MySQL from PHP

This page is designed to give you a very brief overview of accessing a MySQL database from PHP. We will cover the basics of querying tables, retrieving results, updating tables, and error handling.

Overview

While there are many available options for accessing databases from PHP, this tutorial focuses on the PHP-MySQL API. The other APIs (such as ODBC) are similar, and knowledge of one should allow you to easily adapt to others. We will not cover all of the functions available, so be sure to read through the API to understand what functionality is available to you, as well as checking the parameter types and return values for proper error handling.

Establishing a Connection

MySQL is a separate application from your PHP code. The first step to accessing the database system from your program is establishing a connection. To do this, we use the `mysql_connect` function. This may look like the following:

```
$db_connection = mysql_connect("localhost", "cs143", "");
```

This line says to connect to the MySQL database system on the machine "localhost" (on the default port 3306), using the username "cs143" and an empty password. The returned value is a resource handle for the connection, which can be passed to other functions. It is possible to establish more than one connection at a time. When you are finished with a connection, you ***must*** close it with the `mysql_close` function:

```
mysql_close($db_connection);
```

Not closing a connection after you are done is a common mistake that many beginning programmers make. If you don't close connections, you will start getting errors like "connection unavailable" after a while, which will make your Web application inaccssible.

Selecting a Database

Typically after you establish a connection to the MySQL server, you want to specify a particular database which you'll be accessing. This is achieved using the `mysql_select_db` function. To use the database named "TEST", you would issue the following:

```
mysql_select_db("TEST", $db_connection);
```

Issuing Queries

To issue a query (select, update, insert, delete), you can use the `mysql_query` function. To read all of the data from a table named "Student" in the "TEST" database we previously selected, we'd run the following code:

```
$query = "select * from Student";
$rs = mysql_query($query, $db_connection);
```

Retrieving Results

The above query returns a resource which contains the results of the query. We can retrieve the actual values row by row using the function `mysql_fetch_row`. This function returns the column values of the retrieved row in an array, or FALSE if there are no more results. Typically we want to retrieve all results of the query, which can be done with a simple loop:

```
while($row = mysql_fetch_row($rs)) {
    $sid = $row[0];
    $name = $row[1];
    $email = $row[2];
    print "$sid, $name, $email<br />";
}
```

Updating Values

The above example shows a simple select query. Issuing other query types (insert, update, delete) is similar. If we wish to update all email addresses to add ".edu" to the end, we'd issue the following query:

```
$query = "update Student set email = CONCAT(email, '.edu')";
mysql_query($query, $db_connection);
```

To find out how many records were altered by a query, we can use `mysql_affected_rows`:

```
$affected = mysql_affected_rows($db_connection);
print "Total affected rows: $affected<br />";
```

Handling User Input

As a last step, we want to make one very important note: your code should ***never*** trust user input. That is, if you take input from a user (such as in an HTML form) which will be part of your query, you must always "sanitize" their inputs. For example, the user's input may contain escape characters which must be properly handled to avoid errors, or worse, data corruption. Sanitizing user inputs is done with the function `mysql_real_escape_string`. This function takes an input string, and produces a string which has all "special" characters, such as ', properly escaped.

To use user-supplied parameters as part of your query, we typically write the SQL query using C-like printf formatting codes, and then fill in the user supplied data. For example, if we have user input in the variable `$name` which we wish to use in a query, we would do the following:

```
$query = "select * from Student where name = '%s'";
$sanitized_name = mysql_real_escape_string($name, $db_connection);
$query_to_issue = sprintf($query, $sanitized_name);
$rs = mysql_query($query_to_issue, $db_connection);
...
```

Basic Error Handling

Many MySQL functions return resources or boolean types to indicate success and failure. You can use these return values to take proper action when a command fails. Sometimes it may be helpful to get more detailed information on why a function failed. To fetch the MySQL error code and corresponding textual message, PHP-MySQL provides two functions: `mysql_errno` and `mysql_error`. For example, to verify your connection is properly established and exit gracefully when the connection fails, we may write the following:

```
$db_connection = mysql_connect("localhost", "cs143", "");
if(!$db_connection) {
    $errmsg = mysql_error($db_connection);
    print "Connection failed: $errmsg <br />;
    exit(1);
}
...
```