Programming via PHP Database access

7.1. Database fundamentals

One of the most common applications of PHP is to provide a Web interface for accessing a **database**. The database may hold information about user postings for a forum, account and order information for a vendor, or raw data for a statistics site.

Because databases are so common, there are a special category of programs written specifically for managing databases, called **database management systems**(more often referenced by their abbreviation **DBMS**). Typical users don't often know that they're dealing with DBMSs, so the names of specific products aren't widely known. Some of the more popular commercial DBMS packages are Oracle and Microsoft SQL Server, but there are also two prominent open-source DBMS packages, MySQL and PostgreSQL.

We'll use MySQL here, since it is easily available and used quite frequently for Web pages in conjunction with PHP. In fact, PHP's interfaces for interacting with the different DBMSs are all quite similar, so if you wanted to use another, it wouldn't take much to learn.

Most DBMSs represent a database as a set of **tables**, each of which has a relatively fixed set of **columns**, and a more dynamic set of **rows**. A row represents a single entity, and a column represents an attribute of an entity. The simplest thing is to look at an example: Suppose we are managing a Web forum using a database. One thing we might want would be a table listing all participants in the forum. Each participant would have a distinctive user ID, a password, a real name, an e-mail address, and the year the participant joined the forum; each of these attributes will be a column in our table.

`Users` table

`userid`	`passwd`	`name`	`email`	`year_joined`
sherlock	pipe	Sherlock Holmes	221b@bakerst.uk	1878
marple	knitting	Miss Marple	marple@trenton.uk	1923
nancy	magnifier	Nancy Drew	nancy@trenton.us	1958

Of course, the forum database will contain other tables, too. One table it will certainly have would list all posts made in the forum. The following table describes the columns that we will have in this table.

Columns in the `Posts` table

`poster`	the ID of the person posting the message, corresponding to the `userid` field of `Users`

Columns in the `Posts` table

`postdate`	the date and time that the message was posted
`subject`	the subject of the message, as entered by the poster
`body`	the body of the message, as entered by the poster

7.2. Simple SQL retrieval

We will want to ask the DBMS to retrieve information for us. There is a language designed specifically for this, called **SQL** (an acronym for Structured Query Language). SQL is so widely popular that it is closely identified with high-quality DBMSs — which is why so many of the popular DBMSs, including MySQL, incorporate the name SQL into their name.

SQL is a language for composing **queries** for a database. It is not a full-fledged programming language, but it is powerful enough to represent all of the typical operations on a database. PHP scripts send requests to the database using SQL, so we need to learn some of its fundamentals.

For now, we'll look only at the simplest version of the most important SQL command: the **`SELECT` query**. A simple `SELECT` query is composed of three separate clauses: a `SELECT` clause listing the names of the columns whose values we want to see, a `FROM` clause saying which table we want to access, and a `WHERE` clause indicating which rows we want to retrieve from that table.

For example, suppose we wanted to look up Sherlock Holmes' password and e-mail address. The SQL query we would want to send to the DBMS is the following.

```
SELECT passwd, email
FROM Users
WHERE name = 'Sherlock Holmes'
```

Note that the `WHERE` clause is a condition. In this case, we want to compare the `name` of each row to the string Sherlock Holmes (in SQL, strings are enclosed in single quotes). When we find a row that matches, we want the DBMS to send back the columns named in the `SELECT` clause.

The `WHERE` clause can include several conditions joined by `AND` and/or `OR`.

```
SELECT name
FROM Users
WHERE year_joined > 1900 AND year_joined < 1970
```

If the table held the three people listed above, this SQL query would result in two values: Miss Marple and Nancy Drew.

7.3. PHP database functions

PHP provides access to MySQL databases via a number of functions. We'll find six of them particularly useful. They are listed below in the order they will typically be used.

`$db = mysql_connect($dbms_location)`

> Connects to the DBMS whose address is identified by the parameter, which should be a string. The exact string will depend on where exactly the DBMS is located; an example string is `localhost:/export/mysql/mysql.sock`. The function returns a value that can later be used to refer to the DBMS connection.

`mysql_select_db($db_name, $db)`

> Selects which database managed by the DBMS that this PHP connection will reference. The DBMS will have a name for each database, and the name should be passed as a string for the first parameter. The second parameter should be a reference to the DBMS connection as returned by `mysql_connect`.

`$result = mysql_query($sql, $db)`

> Sends a query to the DBMS. The SQL code should be located in the string passed as the first parameter; the second parameter should be a reference to the DBMS connection as returned by `mysql_connect`. The function `mysql_query` returns a value that allows access to whatever the results were of the query; if the query failed — as for example would happen if there were an error in the given SQL — this return value would be `FALSE`.

`mysql_error()`

> If the last executed query failed for some reason, `mysql_error` returns a string that describes this error. I recommend that you always make sure your PHP code somehow echoes the result of `mysql_error` when a query fails, so that you have a way to debug your script.

`mysql_num_rows($result)`

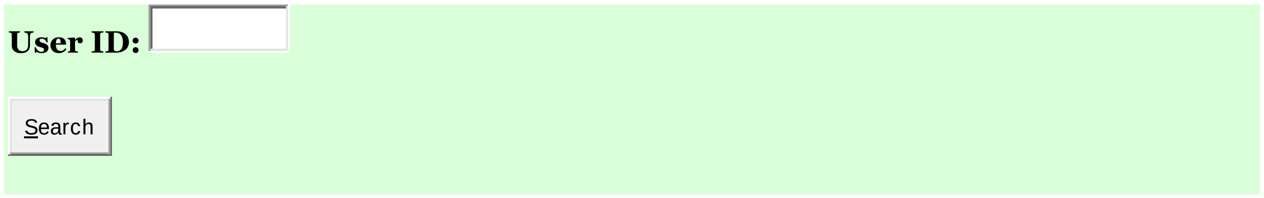
Returns the number of rows found by the SQL query whose results are encapsulated in the parameter. Of course, this could be 0 if there were no rows that resulted from the query.

`mysql_result($result, $row, $col)`

Returns a string holding the information found in row `$row` and column `$col` in the results that are encapsulated by the `$result` parameter. Both rows and columns are numbered starting from 0: That is, the first row is numbered 0, the second row 1, and so on, and similarly the first column is numbered 0.

7.4. Example database access

Let's look at an example where these functions turn out to be useful. Suppose we want to write a script that allows a user of our Web site to view the information about a particular participant. The Web site user would see a form such as the following.



This form can be generated by the following HTML code.

```
<form method="post" action="user.php">
<p><b>User ID:</b> <input type="text" name="userid" /></p>
<p><input type="submit" value="Search" /></p>
</form>
```

The file `user.php` would be the following.

```
<?php
    import_request_variables("pg", "form_");

    $db = mysql_connect("localhost:/export/mysql/mysql.sock");
    mysql_select_db("forum", $db);
    $sql = "SELECT name, email"
        . " FROM Users"
        . " WHERE userid = '$form_userid'";
    $rows = mysql_query($sql, $db);
    if(!$rows) {
        $error = "SQL error: " . mysql_error();
    } elseif(mysql_num_rows($rows) == 0) {
        $error = "No such user name found";
    } else {
        $error = FALSE;
        $user_name = mysql_result($rows, 0, 0);
        $user_email = mysql_result($rows, 0, 1);
    }
?>
<html>
<head>
<title>User information</title>
</head>

<body>
<?php
    if($error) {
        echo "<h1>Error accessing user information</h1>\n";
        echo "<p>$error</p>\n";
    } else {
        echo "<h1>Information about $form_userid</h1>\n";
        echo "<p>User ID: <tt>$form_userid</tt></p>\n";
        echo "<p>Real name: $user_name</p>\n";
        echo "<p>E-mail address: <tt>$user_email</tt></p>\n";
    }
?>
</body>
</html>
```

Notice in particular the SQL generation within the PHP code. This program first constructs a string containing the SQL code, referenced by a variable `$sql`. Because this string is fairly long, and because it is easier to read with each clause on a separate line, the PHP splits the generation of this string across three different lines using the catenation operator. (Notice how there are not semicolons at the end of the first two lines: This is what leads PHP to consider all three lines as part of the same statement, as a semicolon marks the end of the current statement.) Whatever name the user typed in the text field is inserted into the SQL string in the appropriate place.

After executing the query via `mysql_query`, this program first checks whether there is an error, then it verifies that a result was found using `mysql_num_rows`; and if there is a result, then it dumps the result into some variables. The PHP closes the connection and then proceeds to echo information about the result back to the user.

Notice that `mysql_result` in our example retrieves row 0 and column 0 from the result to retrieve the first column of the first row. This numbering scheme isn't the intuitive approach, but it's the way that PHP tends to number things.

We'll see more examples of database access in succeeding chapters, but you'll see that they all use the same template for communicating with the database.

Source: http://www.toves.org/books/php/ch07-db/index.html