Programming via PHP More SQL

9.1. Sorting results

We now return to SQL to get a more thorough spectrum of the types of queries that one can use to access a database. We won't see any PHP concepts in this chapter, though we'll get more practice with what we know. Our primary goal is to extend our SQL knowledge beyond the basic **SELECT** queries that we saw before.

The first extension we'll make is to add one more clause to our **SELECT** query: After the **SELECT**, **FROM**, and the optional **WHERE** clause we can also have an **ORDER BY** clause to specify in what order we want the returned rows to be.

In the previous chapter, we saw how we could list all of the posts in our forum; but MySQL would have the right send them in no particular order, which isn't the behavior we would want. To amend this, we would add an **ORDER BY** clause to our query. (This clause needs to come after the **WHERE** clause; in the case of a query like this without a **WHERE** clause, it belongs after where the **WHERE** clause would be.)

```
SELECT subject, body
FROM Posts
ORDER BY postdate
```

The posts will now be retrieved in order, starting with the earliest post. If we wanted to report the most recent post first, we can do this by adding `DESC` afterwards.

```
SELECT subject, body
FROM Posts
ORDER BY postdate DESC
```

You may list several values in the **ORDER BY** clause, separated by commas. The results will be ordered by the first-mentioned value, then the second-mentioned value would break ties; then the subsequent values.

9.2. Joins

A more sophisticated type of **SELECT** query is the *join query*, where we list multiple tables in its **FROM** clause. We can then draw out information from the various tables.

```
SELECT subject, body, name
FROM Posts, Users
ORDER BY postdate
```

This won't work as you probably expect, however: A simple SQL join will join *every* row from the first table with *every* row from the second table. Thus a post by the `sherlock` user will be joined with each of `sherlock`, `marple`, and `nancy`, each yielding a separate row. If there are 5 posts in the database, and our Web page used the above query, it would list 15 posts, once for each combination of a post with a user. This is certainly not we wanted.

To select only those results where the post and user rows correspond, we need to add that requirement into the **`WHERE`** clause.

```
SELECT subject, body, name
FROM Posts, Users
WHERE poster = userid
ORDER BY postdate
```

This is the query we'll use in our updated version of the page that we saw in the previous chapter for listing all current posts. This page has also been modified to use the **`for`** statement that we saw at the end of the previous chapter.

```
<?php import_request_variables("pg", "form_"); ?>
<html>
<head>
<title>Current Posts</title>
</head>

<body>
<h1>Current Posts</h1>

<?php
    $db = mysql_connect("localhost:/export/mysql/mysql.sock");
    mysql_select_db("forum", $db);
    $sql = "SELECT subject, body, name"
        . " FROM Posts, Users"
        . " WHERE poster = userid"
        . " ORDER BY postdate";
    $rows = mysql_query($sql, $db);
    if(!$rows) {
        echo "<p>SQL error: " . mysql_error() . "</p>\n";
```

```
    } elseif(mysql_num_rows($rows) == 0) {
        echo "<p>There are not yet any posts.</p>\n";
    } else {
        for($row = 0; $row < mysql_num_rows($rows); $rows++) {
            $post_subject = mysql_result($rows, $row, 0);
            $post_body = mysql_result($rows, $row, 1);
            $post_name = mysql_result($rows, $row, 2);

            echo "<h2>$post_subject</h2>\n";
            echo "<p>By: $post_name</p>\n";
            echo "<p>$post_body</p>\n";
        }
    }
?>
</body>
</html>
```

9.3. Inserts

Another common thing one would want to do with a database via the Web is to insert new items into a table. This is done via a different type of SQL query, called an **INSERT** query. As an example of an SQL query, suppose we want a PHP script that adds a new post into the database. The relevant SQL query would be the following.

```
INSERT INTO Posts (poster, postdate, subject, body)
VALUES ('sherlock', '2007-07-04 03:23', 'Case closed', 'The butler did it.')
```

An **INSERT** query has two clauses, the **INSERT INTO** clause and the **VALUES** clause. The **INSERT INTO** clause starts with the name of the table into which we wish to insert (`Posts` here), followed by a set of parentheses enclosing a list of the names of columns whose values we will specify for this new row. The **VALUES** clause consists of a set of parentheses enclosing the values for the columns named in the **INSERT INTO** clause, in the same order. Incidentally, you are allowed to omit most columns from the **INSERT** query, in which case the DBMS will choose some default value for the newly created row.

Unlike a **SELECT** query, an **INSERT** query doesn't really have any results: The information is going into the database, not coming out. As a result, you'd have no reason in your PHP script to use the `mysql_result` function after executing an **INSERT** query.

Let us now look at a PHP script that will execute an **INSERT** query. First, we need to specify the form that the user will complete.

```
<form method="post" action="post.php">
<p>Name: <input type="text" name="user" />
<br />Password: <input type="password" name="passwd" />
<br />Subject: <input type="text" name="subj" />
<br />Body: <input type="text" name="body" />
<br /><input type="submit" value="Post" />
</p></form>
```

Note that we have the user enter a password. Our PHP script will check the password first using a **SELECT** query; if that query finds a row with the relevant user/password combination, then it will proceed to add the post using a **INSERT** query.

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

    $db = mysql_connect("localhost:/export/mysql/mysql.sock");
    mysql_select_db("forum", $db);
    $sql = "SELECT userid"
        . " FROM Users"
        . " WHERE userid = '$form_user' AND passwd = '$form_passwd'";
    $rows = mysql_query($sql, $db);
    if(!$rows) {
        $message = "Password lookup error: " . mysql_error();
    } elseif(mysql_num_rows($rows) == 0) {
        $message = "Password not accepted.";
    } else {
        $sql = "INSERT INTO Posts (poster, postdate, subject, body)"
            . " VALUES ('$form_user', NOW(),"
            . "        '$form_subj', '$form_body')";
        $rows = mysql_query($sql, $db);
        if(!$rows) {
            $message = "Insert error: " . mysql_error();
        } else {
            $message = "The post was successfully inserted.";
        }
    }
?>
<html>
<head>
<title>Post requested</title>
```

```
</head>

<body>
<h1>Post requested</h1>

<p><?php echo $message; ?></p>

<p>[<a href="view.php">List Posts</a>]</p>
</body>
</html>
```

9.4. Other SQL

SQL provides many other types of queries, but they occur less often in PHP scripts than **SELECT** and **INSERT** queries. One that you may find useful, though, is the UPDATEquery, which says to alter existing rows in a table. Let us look at an example that changes the password for a user.

```
UPDATE Users
SET passwd = 'sillyhat'
WHERE userid = 'sherlock'
```

The UPDATE clause specifies which table contains the row we wish to modify; the **SET** clause says how we want to change that row; and the **WHERE** clause provides a condition for identifying the row to modify. In fact, we are permitted to use a **WHERE** clause that matches multiple rows in the table: MySQL will simply apply the **SET**clause for all rows for which the **WHERE** clause holds.

Another thing you occassionally might want to do via PHP is to delete rows from a table. This is done, appropriately enough, with a **DELETE** query, which consists of a**DELETE FROM** clause specifying the table from which we wish to drop rows, and a **WHERE** clause specifying a condition for identifying which rows to drop. The following example will delete all posts from before 2006.

```
DELETE FROM Posts
WHERE postdate < '2006-01-01'
```

Of course, you would want to be careful with a **DELETE** query, because if you mess up your **WHERE** clause you could end up clobbering the entire table. Generally speaking, PHP scripts won't delete information: The database will only accumulate information.

SQL provides many other query types, but they are targeted toward managing a database, such as creating tables, or renaming a column, or adding new columns to an existing table.

Source: http://www.toves.org/books/php/ch09-moresql/index.html