Retrieving Query Results

The preceding section of this chapter demonstrates how to execute simple queries on a MySQL database. A *simple query*, as I'm calling it, could be defined as one that begins with `INSERT`, `UPDATE`, `DELETE`, or `ALTER`. What all four of these have in common is that they return no data, just an indication of their success. Conversely, a `SELECT` query generates information (i.e., it will return rows of records) that has to be handled by other PHP functions.

The primary tool for handling `SELECT` query results is `mysqli_fetch_array()`, which uses the query result variable (that I've been calling `$r`) and returns one row of data at a time, in an array format. You'll want to use this function within a loop that will continue to access every returned row as long as there are more to be read. The basic construction for reading every record from a query is

```
while ($row = mysqli_fetch_array($r)) {
    // Do something with $row.
}
```

You will almost always want to use a `while` loop to fetch the results from a `SELECT` query.

The `mysqli_fetch_array()` function takes an optional second parameter specifying what type of array is returned: associative, indexed, or both. An associative array allows you to refer to column values by name, whereas an indexed array requires you to use only numbers (starting at 0 for the first column returned). Each parameter is defined by a constant listed in **Table 8.1**. The `MYSQLI_NUM` setting is marginally faster (and uses less memory) than the other options. Conversely, `MYSQLI_ASSOC` is more overt (`$row['column']` rather than `$row[3]`) and may continue to work even if the query changes.

An optional step you can take when using `mysqli_fetch_array()` would be to free up the query result resources once you are done using them:

```
mysqli_free_result ($r);
```

This line removes the overhead (memory) taken by `$r`. It's an optional step, since PHP will automatically free up the resources at the end of a script, but—like using `mysqli_close()`—it does make for good programming form.

To demonstrate how to handle results returned by a query, let's create a script for viewing all of the currently registered users.

Table 8.1 Adding one of these constants as an optional parameter to the `mysqli_fetch_array()` function dictates how you can access the values returned. The default setting of the function is `MYSQLI_BOTH`.

mysqli_fetch_array() Constants	
Constant	**Example**
`MYSQLI_ASSOC`	`$row['column']`
`MYSQLI_NUM`	`$row[0]`
`MYSQLI_BOTH`	`$row[0]` or `$row['column']`

To retrieve query results:

1. Create a new PHP document in your text editor or IDE (**Script 8.4**).

```
<?php # Script 8.4 - view_users.php

$page_title = 'View the Current Users';

include ('includes/header.html');

echo '<h1>Registered Users</h1>';
```

2. Connect to and query the database.

```
require_once
→ ('../mysqli_connect.php');

$q = "SELECT CONCAT(last_name, ', ',
→ first_name) AS name,
→ DATE_FORMAT(registration_date, '%M
→ %d, %Y') AS dr FROM users ORDER BY
→ registration_date ASC";

$r = @mysqli_query ($dbc, $q);
```

The query here will return two columns (**Figure 8.12**): the users' names (formatted as *Last Name, First Name*) and the date they registered (formatted as *Month DD, YYYY*). Because both columns are formatted using MySQL functions, aliases are given to the returned results (*name* and *dr*, accordingly). See Chapter 5 if you are confused by any of this syntax.

3. Display the query results.

```
if ($r) {

    echo '<table align="center"
    → cellspacing="3"
    cellpadding="3"
    → width="75%">

    <tr><td
    → align="left"><b>Name</b></td><
    td
    → align="left"><b>Date
    → Registered</b></td></tr>

';
```

Script 8.4 The `view_users.php` script runs a static query on the database and prints all of the returned rows.

Script

```
1   <?php # Script 8.4 - view_users.php

2   // This script retrieves all the records
    from the users table.

3

4   $page_title = 'View the Current Users';

5   include ('includes/header.html');

6

7   // Page header:

8   echo '<h1>Registered Users</h1>';

9

10  require_once ('../mysqli_connect.php'); //
    Connect to the db.

11

12  // Make the query:

13  $q = "SELECT CONCAT(last_name, ', ',
    first_name) AS name,
    DATE_FORMAT(registration_date, '%M %d,
    %Y') AS dr FROM users ORDER BY
    registration_date ASC";

14  $r = @mysqli_query ($dbc, $q); // Run the
    query.

15

16  if ($r) { // If it ran OK, display the
    records.

17

18    // Table header.

19    echo '<table align="center"
      cellspacing="3" cellpadding="3"
      width="75%">

20    <tr><td align="left"><b>Name</b></td><td
      align="left"><b>Date
      Registered</b></td></tr>

21  ';

22

23    // Fetch and print all the records:

24    while ($row = mysqli_fetch_array($r,
      MYSQLI_ASSOC)) {
```

(script continues on next page)

Script 8.4 *continued*

```
Script
25        echo '<tr><td align="left">' .
          $row['name'] . '</td><td align="left">'
          . $row['dr'] . '</td></tr>
26      ';
27      }
28
29      echo '</table>'; // Close the table.
30
31      mysqli_free_result ($r); // Free up the
        resources.
32
33    } else { // If it did not run OK.
34
35      // Public message:
36      echo '<p class="error">The current users
        could not be retrieved. We apologize for
        any inconvenience.</p>';
37
38      // Debugging message:
39      echo '<p>' . mysqli_error($dbc) . '<br
        /><br />Query: ' . $q . '</p>';
40
41    } // End of if ($r) IF.
42
43    mysqli_close($dbc); // Close the database
      connection.
44
45    include ('includes/footer.html');
46    ?>
```

```
while ($row =
→ mysqli_fetch_array($r,
→ MYSQLI_ASSOC)) {
  echo '<tr><td align="left">' .
  → $row['name'] . '</td><td
  → align="left">' . $row['dr'] .
  → '</td></tr>
        ';
}
echo '</table>';
```

To display the results, make a table and a header row in HTML. Then loop through the results using `mysqli_fetch_array()` and print each fetched row. Finally, close the table.

Notice that within the `while` loop, the code refers to each returned value using the proper alias: `$row['name']` and `$row['dr']`. The script could not refer to `$row['first_name']` or `$row['date_registered']` because no such field name was returned (see Figure 8.12).

continues on next page

```
Terminal
mysql> SELECT CONCAT(last_name, ', ', first_name) AS
name, DATE_FORMAT(registration_date, '%M %d, %Y') AS
dr FROM users ORDER BY registration_date ASC;
+--------------------+--------------------+
| name               | dr                 |
+--------------------+--------------------+
| Ullman, Larry      | September 22, 2007 |
| Isabella, Zoe      | September 22, 2007 |
| Starr, Ringo       | September 22, 2007 |
| Harrison, George   | September 22, 2007 |
| McCartney, Paul    | September 22, 2007 |
| Lennon, John       | September 22, 2007 |
| Brautigan, Richard | September 22, 2007 |
| Banks, Russell     | September 22, 2007 |
| Simpson, Homer     | September 22, 2007 |
| Simpson, Marge     | September 22, 2007 |
| Simpson, Bart      | September 22, 2007 |
| Simpson, Lisa      | September 22, 2007 |
| Simpson, Maggie    | September 22, 2007 |
| Simpson, Abe       | September 22, 2007 |
| Chabon, Michael    | September 22, 2007 |
| Greene, Graham     | September 22, 2007 |
| DeLillo, Don       | September 22, 2007 |
| Jones, David       | September 22, 2007 |
| Dolenz, Micky      | September 22, 2007 |
| Nesmith, Mike      | September 22, 2007 |
| Sedaris, David     | September 22, 2007 |
| Hornby, Nick       | September 22, 2007 |
| Bank, Melissa      | September 22, 2007 |
| Morrison, Toni     | September 22, 2007 |
| Franzen, Jonathan  | September 22, 2007 |
| Campbell, Bob      | September 30, 2007 |
+--------------------+--------------------+
26 rows in set (0.00 sec)

mysql>
```

Figure 8.12 The query results as run within the mysql client.

4. Free up the query resources.

```
mysqli_free_result ($r);
```

Again, this is an optional step but a good one to take.

5. Complete the main conditional.

```
} else {
    echo '<p class="error">The
    → current users could not be
    → retrieved. We apologize for
    any
    → inconvenience.</p>';
    echo '<p>' . mysqli_error($dbc)
    .
    → '<br /><br />Query: ' . $q .
    → '</p>';
}
```

As in the `register.php` example, there are two kinds of error messages here. The first is a generic message, the type you'd show in a live site. The second is much more detailed, printing both the MySQL error and the query, both being critical for debugging purposes.

6. Close the database connection and finish the page.

```
mysqli_close($dbc);
include ('includes/footer.html');
?>
```

7. Save the file as `view_users.php`, place it in your Web directory, and test it in your browser (**Figure 8.13**).

✔ Tips

- The function `mysqli_fetch_row()`is the equivalent of `mysqli_fetch_array ($r, MYSQLI_NUM);`
- The function `mysqli_fetch_assoc()` is the equivalent of `mysqli_fetch_array ($r, MYSQLI_ASSOC);`
- As with any associative array, when you retrieve records from the database, you must refer to the columns exactly as they are defined in the database. This is to say that the keys are case-sensitive.
- If you are in a situation where you need to run a second query inside of your `while` loop, be certain to use different variable names for that query. For example, the inner query would use `$r2` and `$row2` instead of `$r` and `$row`. If you don't do this, you'll encounter logical errors.
- I frequently see beginning PHP developers muddle the process of fetching query results. Remember that you must execute the query using `mysqli_query()`, and then use `mysqli_fetch_array()` to retrieve a single row of information. If you have multiple rows to retrieve, use a `while` loop.

Registered Users

Name	Date Registered
Ullman, Larry	September 22, 2007
Isabella, Zoe	September 22, 2007
Starr, Ringo	September 22, 2007
Harrison, George	September 22, 2007
McCartney, Paul	September 22, 2007
Lennon, John	September 22, 2007
Brautigan, Richard	September 22, 2007
Banks, Russell	September 22, 2007
Simpson, Homer	September 22, 2007
Simpson, Marge	September 22, 2007
Simpson, Bart	September 22, 2007
Simpson, Lisa	September 22, 2007
Simpson, Maggie	September 22, 2007
Simpson, Abe	September 22, 2007
Chabon, Michael	September 22, 2007
Greene, Graham	September 22, 2007
DeLillo, Don	September 22, 2007
Jones, David	September 22, 2007
Dolenz, Micky	September 22, 2007
Nesmith, Mike	September 22, 2007
Sedaris, David	September 22, 2007
Hornby, Nick	September 22, 2007
Bank, Melissa	September 22, 2007
Morrison, Toni	September 22, 2007
Franzen, Jonathan	September 22, 2007
Campbell, Bob	September 30, 2007

Figure 8.13 All of the user records are retrieved from the database and displayed in the Web browser.

Ensuring Secure SQL

Database security with respect to PHP comes down to three broad issues:

1. Protecting the MySQL access information
2. Not revealing too much about the database
3. Being cautious when running queries, particularly those involving user-submitted data

You can accomplish the first objective by securing the MySQL connection script outside of the Web directory so that it is never viewable through a Web browser (see Figure 8.3). I discuss this in some detail earlier in the chapter. The second objective is attained by not letting the user see PHP's error messages or your queries (in these scripts, that information is printed out for your debugging purposes; you'd never want to do that on a live site).

For the third objective, there are numerous steps you can and should take, all based upon the premise of never trusting user-supplied data. First, validate that some value has been submitted, or that it is of the proper type (number, string, etc.). Second, use regular expressions to make sure that submitted data matches what you would expect it to be (this topic is covered in Chapter 13, "Perl-Compatible Regular Expressions"). Third, you can typecast some values to guarantee that they're numbers (discussed in Chapter 12, "Security Methods"). A fourth recommendation is to run user-submitted data through the `mysqli_real_escape_string()` function. This function cleans data by escaping what could be problematic characters. It's used like so:

```
$clean = mysqli_real_escape_string($dbc,
→ data);
```

For security purposes, `mysqli_real_escape_string()` should be used on every text input in a form. To demonstrate this, let's revamp `register.php` (Script 8.3).

To use mysqli_real_escape_string():

1. Open `register.php` (Script 8.3) in your text editor or IDE.
2. Move the inclusion of the `mysqli_connect.php` file (line 46 in Script 8.3) to just after the main conditional (**Script 8.5**).

   Because the `mysqli_real_escape_string()` function requires a database connection, the `mysqli_connect.php` script must be required earlier in the script.

Script 8.5 The `register.php` script now uses the `mysqli_real_escape_string()` function to clean the submitted data.

Script

```
1   <?php # Script 8.5 - register.php #2
2
3   $page_title = 'Register';
4   include ('includes/header.html');
5
6   // Check if the form has been submitted:
7   if (isset($_POST['submitted'])) {
8
9     require_once ('../mysqli_connect.php');
      // Connect to the db.
10
11    $errors = array(); // Initialize an
      error array.
12
13    // Check for a first name:
14    if (empty($_POST['first_name'])) {
15       $errors[] = 'You forgot to enter your
         first name.';
16    } else {
17       $fn = mysqli_real_escape_string($dbc,
         trim($_POST['first_name']));
18    }
19
20    // Check for a last name:
21    if (empty($_POST['last_name'])) {
22       $errors[] = 'You forgot to enter your
         last name.';
23    } else {
24       $ln = mysqli_real_escape_string($dbc,
         trim($_POST['last_name']));
25    }
26
27    // Check for an email address:
28    if (empty($_POST['email'])) {
29       $errors[] = 'You forgot to enter your
         email address.';
30    } else {
31       $e = mysqli_real_escape_string($dbc,
         trim($_POST['email']));
```

(script continues on next page)

Script 8.5 *continued*

```
}

// Check for a password and match against the confirmed password:
if (!empty($_POST['pass1'])) {
   if ($_POST['pass1'] != $_POST['pass2']) {
     $errors[] = 'Your password did not match the confirmed password.';
   } else {
     $p = mysqli_real_escape_string($dbc, trim($_POST['pass1']));
   }
} else {
   $errors[] = 'You forgot to enter your password.';
}

if (empty($errors)) { // If everything's OK.

   // Register the user in the database...

   // Make the query:
   $q = "INSERT INTO users (first_name, last_name, email, pass, registration_date) VALUES ('$fn', '$ln', '$e', SHA1('$p'), NOW() )";
   $r = @mysqli_query ($dbc, $q); // Run the query.
   if ($r) { // If it ran OK.

     // Print a message:
     echo '<h1>Thank you!</h1>
   <p>You are now registered. In Chapter 11 you will actually be able to log in!</p><p><br /></p>';

   } else { // If it did not run OK.

     // Public message:
```

(script continues on next page)

3. Change the validation routines to use the `mysqli_real_escape_string()` function, replacing each occurrence of `$var = trim($_POST['var'])` with `$var = mysqli_real_escape_string($dbc, trim($_POST['var']))`.

```
$fn = mysqli_real_escape_string($dbc, trim($_POST['first_name']));
$ln = mysqli_real_escape_string($dbc, trim($_POST['last_name']));
$e = mysqli_real_escape_string($dbc, trim($_POST['email']));
$p = mysqli_real_escape_string($dbc, trim($_POST['pass1']));
```

Instead of just assigning the submitted value to each variable (`$fn`, `$ln`, etc.), the values will be run through the `mysqli_real_escape_string()` function first. The `trim()` function is still used to get rid of any unnecessary spaces.

continues on next page

4. Add a second call to `mysqli_close()` before the end of the main conditional.

   ```
   mysqli_close($dbc);
   ```

   To be consistent, since the database connection is opened as the first step of the main conditional, it should be closed as the last step of this same conditional. It still needs to be closed before including the footer and terminating the script (lines 72 and 73), though.

Script 8.5 *continued*

Script

```
61          echo '<h1>System Error</h1>
62          <p class="error">You could not be
            registered due to a system error.
            We apologize for any
            inconvenience.</p>';
63
64          // Debugging message:
65          echo '<p>' . mysqli_error($dbc) .
            '<br /><br />Query: ' . $q .
            '</p>';
66
67        } // End of if ($r) IF.
68
69        mysqli_close($dbc); // Close the
          database connection.
70
71        // Include the footer and quit the
          script:
72        include ('includes/footer.html');
73        exit();
74
75      } else { // Report the errors.
76
77        echo '<h1>Error!</h1>
78        <p class="error">The following
          error(s) occurred:<br />';
79        foreach ($errors as $msg) { // Print
          each error.
80          echo " - $msg<br />\n";
81        }
82        echo '</p><p>Please try
          again.</p><p><br /></p>';
83
84      } // End of if (empty($errors)) IF.
85
86      mysqli_close($dbc); // Close the
        database connection.
87
88    } // End of the main Submit conditional.
89    ?>
```

(script continues on next page)

Script 8.5 *continued*

```
<h1>Register</h1>
<form action="register.php" method="post">
  <p>First Name: <input type="text" name="first_name" size="15" maxlength="20" value="<?php if (isset($_POST['first_name'])) echo $_POST['first_name']; ?>" /></p>
  <p>Last Name: <input type="text" name="last_name" size="15" maxlength="40" value="<?php if (isset($_POST['last_name'])) echo $_POST['last_name']; ?>" /></p>
  <p>Email Address: <input type="text" name="email" size="20" maxlength="80" value="<?php if (isset($_POST['email'])) echo $_POST['email']; ?>"  /> </p>
  <p>Password: <input type="password" name="pass1" size="10" maxlength="20" /></p>
  <p>Confirm Password: <input type="password" name="pass2" size="10" maxlength="20" /></p>
  <p><input type="submit" name="submit" value="Register" /></p>
  <input type="hidden" name="submitted" value="TRUE" />
</form>
<?php
include ('includes/footer.html');
?>
```

5. Save the file as `register.php`, place it in your Web directory, and test it in your Web browser (**Figures 8.14** and **8.15**).

continues on next page

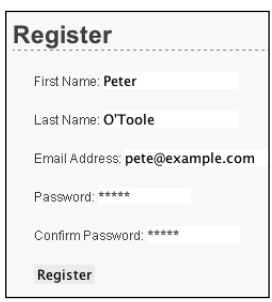


Figure 8.14 Values with apostrophes in them, like a person's last name, will no longer break the `INSERT` query, thanks to the `mysqli_real_escape_string()` function.

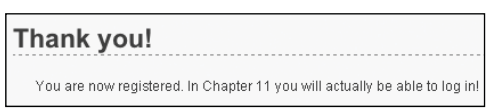


Figure 8.15 Now the registration process will handle problematic characters and be more secure.

✔ Tips

- The `mysqli_real_escape_string()` function escapes a string in accordance with the language being used, which is an added advantage over alternative solutions.
- If you see results like those in **Figure 8.16**, it means that the `mysqli_real_escape_string()` function cannot access the database (because it has no connection, like `$dbc`).
- If Magic Quotes is enabled on your server (which means you're using a version of PHP prior to 6), you'll need to remove any slashes added by Magic Quotes, prior to using the `mysqli_real_escape_string()` function. The code (cumbersome as it is) would look like:

```
$fn = mysqli_real_escape_string
→ ($dbc, trim (stripslashes
→ ($_POST['first_name'])));
```

If you don't use `stripslashes()` and Magic Quotes is enabled, the form values will be doubly escaped.

Notice: Undefined variable: dbc in **/Applications/Abyss Web Server/htdocs/register.php** on line **17**

Warning: mysqli_real_escape_string() expects parameter 1 to be mysqli, null given in **/Applications/Abyss Web Server/htdocs/register.php** on line **17**

Figure 8.16 Since the `mysqli_real_escape_string()` requires a database connection, using it without that connection (e.g., before including the connection script) can lead to other errors.

Modifying register.php

The `mysqli_num_rows()` function could be applied to `register.php` to prevent someone from registering with the same email address multiple times. Although the `UNIQUE` index on that column in the database will prevent that from happening, such attempts will create a MySQL error. To prevent this using PHP, run a `SELECT` query to confirm that the email address isn't currently registered. That query would be simply

```
SELECT user_id FROM users WHERE email='$e'
```

You would run this query (using the `mysqli_query()` function) and then call `mysqli_num_rows()`. If `mysqli_num_rows()` returns 0, you know that the email address hasn't already been registered and it's safe to run the `INSERT`.

Script 8.6 Now the `view_users.php` script will display the total number of registered users, thanks to the `mysqli_num_rows()` function.

Script

```
<?php # Script 8.6 - view_users.php #2
// This script retrieves all the records from the users table.

$page_title = 'View the Current Users';
include ('includes/header.html');

// Page header:
echo '<h1>Registered Users</h1>';

require_once ('../mysqli_connect.php'); // Connect to the db.

// Make the query:
$q = "SELECT CONCAT(last_name, ', ', first_name) AS name, DATE_FORMAT(registration_date, '%M %d, %Y') AS dr FROM users ORDER BY registration_date ASC";
$r = @mysqli_query ($dbc, $q); // Run the query.

// Count the number of returned rows:
$num = mysqli_num_rows($r);

if ($num > 0) { // If it ran OK, display the records.

  // Print how many users there are:
  echo "<p>There are currently $num registered users.</p>\n";

  // Table header.
  echo '<table align="center" cellspacing="3" cellpadding="3" width="75%">
```

(script continues on next page)

Counting Returned Records

The next logical function to discuss is `mysqli_num_rows()`. This function returns the number of rows retrieved by a `SELECT` query. It takes one argument, the query result variable:

```
$num = mysqli_num_rows($r);
```

Although simple in purpose, this function is very useful. It's necessary if you want to paginate your query results (an example of this can be found in the next chapter). It's also a good idea to use this function before you attempt to fetch any results using a `while` loop (because there's no need to fetch the results if there aren't any, and attempting to do so may cause errors). In this next sequence of steps, let's modify `view_users.php` to list the total number of registered users. For another example of how you might use `mysqli_num_rows()`, see the sidebar.

To modify view_users.php:

1. Open `view_users.php` (refer to Script 8.4) in your text editor or IDE.
2. Before the `if ($r)` conditional, add this line (**Script 8.6**)

   ```
   $num = mysqli_num_rows ($r);
   ```

   This line will assign the number of rows returned by the query to the `$num` variable.
3. Change the original `$r` conditional to

   ```
   if ($num > 0) {
   ```

   The conditional as it was written before was based upon whether the query did or did not successfully run, not whether or not any records were returned. Now it will be more accurate.

continues on next page

4. Before creating the HTML table, print the number of registered users.

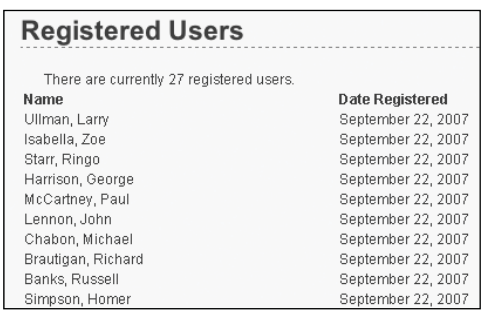
```
echo "<p>There are currently $num
→ registered users.</p>\n";
```

5. Change the `else` part of the main conditional to read

```
echo '<p class="error">There are
→ currently no registered users.</p>';
```

The original conditional was based upon whether or not the query worked. Hopefully you've successfully debugged the query so that it is working and the original error messages are no longer needed. Now the error message just indicates if no records were returned.

6. Save the file as `view_users.php`, place it in your Web directory, and test it in your Web browser (**Figure 8.17**).



Registered Users

There are currently 27 registered users.

Name	Date Registered
Ullman, Larry	September 22, 2007
Isabella, Zoe	September 22, 2007
Starr, Ringo	September 22, 2007
Harrison, George	September 22, 2007
McCartney, Paul	September 22, 2007
Lennon, John	September 22, 2007
Chabon, Michael	September 22, 2007
Brautigan, Richard	September 22, 2007
Banks, Russell	September 22, 2007
Simpson, Homer	September 22, 2007

Figure 8.17 The number of registered users is now displayed at the top of the page.

Script 8.6 *continued*

Script

```
  <tr><td align="left"><b>Name</b></td><td
align="left"><b>Date
Registered</b></td></tr>
';

  // Fetch and print all the records:
  while ($row = mysqli_fetch_array($r,
MYSQLI_ASSOC)) {
  echo '<tr><td align="left">' .
$row['name'] . '</td><td align="left">' .
$row['dr'] . '</td></tr>
  ';
  }

  echo '</table>'; // Close the table.

  mysqli_free_result ($r); // Free up the
resources.

} else { // If no records were returned.

  echo '<p class="error">There are
  currently no registered users.</p>';

}

mysqli_close($dbc); // Close the database
connection.

include ('includes/footer.html');
?>
```

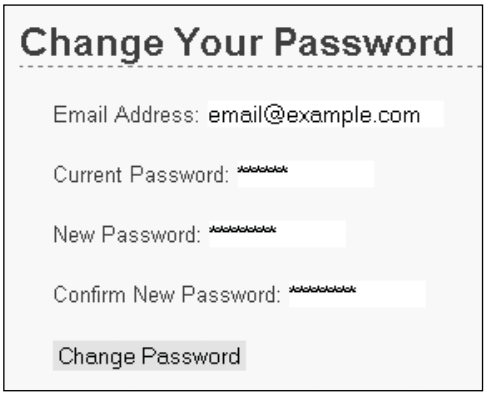


Figure 8.18 The form for changing a user's password.

Updating Records with PHP

The last technique in this chapter shows how to update database records through a PHP script. Doing so requires an `UPDATE` query, and its successful execution can be verified with PHP's `mysqli_affected_rows()` function.

While the `mysqli_num_rows()` function will return the number of rows generated by a `SELECT` query, `mysqli_affected_rows()` returns the number of rows affected by an `INSERT`, `UPDATE`, or `DELETE` query. It's used like so:

```
$num = mysqli_affected_rows($dbc);
```

Unlike `mysqli_num_rows()`, the one argument the function takes is the database connection (`$dbc`), not the results of the previous query (`$r`).

The following example will be a script that allows registered users to change their password. It demonstrates two important ideas:

- Checking a submitted username and password against registered values (the key to a login system as well)
- Updating database records using the primary key as a reference

As with the registration example, this one PHP script will both display the form (**Figure 8.18**) and handle it.

To update records with PHP:

1. Create a new PHP script in your text editor or IDE (**Script 8.7**).

```
<?php # Script 8.7 - password.php
$page_title = 'Change Your Password';
include ('includes/header.html');
```

2. Start the main conditional.

```
if (isset($_POST['submitted'])) {
```

Since this page both displays and handles the form, it'll use the standard conditional.

3. Include the database connection and create an array for storing errors.

```
require_once ('../mysqli_connect.php');
$errors = array();
```

The initial part of this script mimics the registration form.

Script 8.7 The `password.php` script runs an `UPDATE` query on the database and uses the `mysqli_affected_rows()` function to confirm the change.

Script

```
1   <?php # Script 8.7 - password.php
2   // This page lets a user change their
    password.
3
4   $page_title = 'Change Your Password';
5   include ('includes/header.html');
6
7   // Check if the form has been submitted:
8   if (isset($_POST['submitted'])) {
9
10    require_once ('../mysqli_connect.php');
      // Connect to the db.
11
12    $errors = array(); // Initialize an
      error array.
13
14    // Check for an email address:
15    if (empty($_POST['email'])) {
16       $errors[] = 'You forgot to enter your
         email address.';
17    } else {
18       = mysqli_real_escape_string($dbc,
         trim($_POST['email']));
19    }
20
21    // Check for the current password:
22    if (empty($_POST['pass'])) {
23       $errors[] = 'You forgot to enter your
         current password.';
24    } else {
25       $p = mysqli_real_escape_string($dbc,
         trim($_POST['pass']));
26    }
27
28    // Check for a new password and match
29    // against the confirmed password:
```

(script continues on next page)

Script 8.7 *continued*

Script

```
if (!empty($_POST['pass1'])) {
   if ($_POST['pass1'] !=
   $_POST['pass2']) {
     $errors[] = 'Your new password did
     not match the confirmed password.';
   } else {
     $np =
     mysqli_real_escape_string($dbc,
     trim($_POST['pass1']));
   }
} else {
   $errors[] = 'You forgot to enter your
   new password.';
}

if (empty($errors)) { // If everything's
OK.

   // Check that they've entered the
   right email address/password
   combination:
   $q = "SELECT user_id FROM users WHERE
   (email='$e' AND pass=SHA1('$p') )";
   $r = @mysqli_query($dbc, $q);
   $num = @mysqli_num_rows($r);
   if ($num == 1) { // Match was made.

     // Get the user_id:
     $row = mysqli_fetch_array($r,
     MYSQLI_NUM);

     // Make the UPDATE query:
     $q = "UPDATE users SET
     pass=SHA1('$np') WHERE
     user_id=$row[0]";
     $r = @mysqli_query($dbc, $q);

     if (mysqli_affected_rows($dbc) ==
     1) { // If it ran OK.

```

(script continues on next page)

4. Validate the email address and current password fields.

```
if (empty($_POST['email'])) {
    $errors[] = 'You forgot to enter
    → your email address.';
} else {
    $e =
    → mysqli_real_escape_string($dbc
    ,
    → trim($_POST['email']));
}
if (empty($_POST['pass'])) {
    $errors[] = 'You forgot to enter
    → your current password.';
} else {
    $p =
    → mysqli_real_escape_string($dbc
    ,
    → trim($_POST['pass']));
}
```

The form (Figure 8.18) has four inputs: the email address, the current password, and two for the new password. The process for validating each of these is the same as it is in `register.php`. Any data that passes the validation test will be trimmed and run through the `mysqli_real_escape_string()` function, so that it is safe to use in a query.

continues on next page

5. Validate the new password.

```
if (!empty($_POST['pass1'])) {
    if ($_POST['pass1'] !=
    → $_POST['pass2']) {
      $errors[] = 'Your new password
      → did not match the confirmed
      → password.';
    } else {
      $np =
      → mysqli_real_escape_string($
      → dbc, trim($_POST['pass1']));
    }
} else {
    $errors[] = 'You forgot to enter
    → your new password.';
}
```

This code is also exactly like that in the registration script, except that a valid new password is assigned to a variable called `$np` (because `$p` represents the current password).

Script 8.7 *continued*

Script

```
                // Print a message.
                echo '<h1>Thank you!</h1>
                <p>Your password has been
                updated. In Chapter 11 you
                will actually be able to log
                in!</p><p><br /></p>';

        } else { // If it did not run OK.

                // Public message:
                echo '<h1>System Error</h1>
                <p class="error">Your
                password could not be
                changed due to a system
                error. We apologize for any
                inconvenience.</p>';

                // Debugging message:
                echo '<p>' .
                mysqli_error($dbc) . '<br
                /><br />Query: ' . $q .
                '</p>';

        }

        // Include the footer and quit the
        script (to not show the form).
        include ('includes/footer.html');
        exit();

      } else { // Invalid email
      address/password combination.
        echo '<h1>Error!</h1>
        <p class="error">The email address
        and password do not match those on
        file.</p>';
      }

    } else { // Report the errors.
```

(script continues on next page)

Script 8.7 *continued*

```
82
83        echo '<h1>Error!</h1>
84        <p class="error">The following
          error(s) occurred:<br />';
85        foreach ($errors as $msg) { // Print
          each error.
86          echo " - $msg<br />\n";
87        }
88        echo '</p><p>Please try
          again.</p><p><br /></p>';
89
90     } // End of if (empty($errors)) IF.
91
92     mysqli_close($dbc); // Close the
       database connection.
93
94   } // End of the main Submit conditional.
95   ?>
96   <h1>Change Your Password</h1>
97   <form action="password.php" method="post">
98     <p>Email Address: <input type="text"
       name="email" size="20" maxlength="80"
       value="<?php if (isset($_POST['email']))
       echo $_POST['email']; ?>"  /> </p>
99     <p>Current Password: <input
       type="password" name="pass" size="10"
       maxlength="20" /></p>
100    <p>New Password: <input type="password"
       name="pass1" size="10" maxlength="20"
       /></p>
101    <p>Confirm New Password: <input
       type="password" name="pass2" size="10"
       maxlength="20" /></p>
102    <p><input type="submit" name="submit"
       value="Change Password" /></p>
103    <input type="hidden" name="submitted"
       value="TRUE" />
104  </form>
105  <?php
106  include ('includes/footer.html');
107  ?>
```

6. If all the tests are passed, retrieve the user's ID.

```
if (empty($errors)) {
    $q = "SELECT user_id FROM users
    → WHERE (email='$e' AND
    → pass=SHA1('$p') )";
    $r = @mysqli_query($dbc, $q);
    $num = @mysqli_num_rows($r);
    if ($num == 1) {
      $row = mysqli_fetch_array($r,
      → MYSQLI_NUM);
```

This first query will return just the `user_id` field for the record that matches the submitted email address and password (**Figure 8.19**). To compare the submitted password against the stored one, encrypt it again with the `SHA1()` function. If the user is registered and has correctly entered both the email address and password, exactly one row will be selected (since the email value must be unique across all rows). Finally, this one record is assigned as an array (of one element) to the `$row` variable.

continues on next page

```
mysql> SELECT user_id FROM users
WHERE (email='email@example.com'
AND pass=SHA1('mypass') );
+---------+
| user_id |
+---------+
|       1 |
+---------+
1 row in set (0.02 sec)

mysql>
```

Figure 8.19 The result when running the `SELECT` query from the script (the first of two queries it has) within the mysql client.

If this part of the script doesn't work for you, apply the standard debugging methods: remove the error suppression operators (@) so that you can see what errors, if any, occur; use the `mysqli_error()` function to report any MySQL errors; and print, then run the query using another interface (as in Figure 8.19).

7. Update the database.

```
$q = "UPDATE users SET
→ pass=SHA1('$np') WHERE
→ user_id=$row[0]";
$r = @mysqli_query($dbc, $q);
```

This query will change the password—using the new submitted value—where the `user_id` column is equal to the number retrieved from the previous query.

8. Check the results of the query.

```
if (mysqli_affected_rows($dbc) == 1) {
    echo '<h1>Thank you!</h1>
    <p>Your password has been
    → updated. In Chapter 11 you
    will
    → actually be able to log
    → in!</p><p><br /></p>';
} else {
    echo '<h1>System Error</h1>
    <p class="error">Your password
    → could not be changed due to a
    → system error. We apologize for
    → any inconvenience.</p>';
    echo '<p>' . mysqli_error($dbc)
    .
    → '<br /><br />Query: ' . $q .
    → '</p>';
}
```

This part of the script again works similar to `register.php`. In this case, if `mysqli_affected_rows()` returns the number *1*, the record has been updated, and a success message will be printed. If not, both a public, generic message and a more useful debugging message will be printed.

9. Include the footer and terminate the script.

```
include ('includes/footer.html');
exit();
```

At this point in the script, the `UPDATE` query has been run. It either worked or it did not (because of a system error). In both cases, there's no need to show the form again, so the footer is included (to complete the page) and the script is terminated, using the `exit()` function.

10. Complete the `if ($num == 1)` conditional.

```
} else {
    echo '<h1>Error!</h1>
    <p class="error">The email
    → address and password do not
    → match those on file.</p>';
}
```

If `mysqli_num_rows()` does not return a value of *1*, then the submitted email address and password do not match those on file and this error is printed. In this case, the form will be displayed again so that the user can enter the correct information.

11. Print any validation error messages.

```
} else {
    echo '<h1>Error!</h1>
    <p class="error">The following
    → error(s) occurred:<br />';
    foreach ($errors as $msg) {
        echo " - $msg<br />\n";
    }
    echo '</p><p>Please try
    → again.</p><p><br /></p>';
}
```

This `else` clause applies if the `$errors` array is not empty (which means that the form data did not pass all the validation tests). As in the registration page, the errors will be printed.

12. Close the database connection and complete the PHP code.

```
    mysqli_close($dbc);
}
?>
```

13. Display the form.

```
<h1>Change Your Password</h1>
<form action="password.php"
→ method="post">
    <p>Email Address: <input
    → type="text" name="email"
    → size="20" maxlength="80"
    → value="<?php if
    → (isset($_POST['email'])) echo
    → $_POST['email']; ?>" /> </p>
    <p>Current Password: <input
    → type="password" name="pass"
    → size="10" maxlength="20" /></p>
    <p>New Password: <input
    → type="password" name="pass1"
    → size="10" maxlength="20" /></p>
    <p>Confirm New Password: <input
    → type="password" name="pass2"
    → size="10" maxlength="20" /></p>
    <p><input type="submit"
    → name="submit" value="Change
    → Password" /></p>
    <input type="hidden"
    → name="submitted" value="TRUE"
    → />
</form>
```

The form takes three different inputs of type password—the current password, the new one, and a confirmation of the new password—and one text input for the email address. The email address input is sticky (password inputs cannot be).

continues on next page

14. Include the footer file.

```
<?php
include ('includes/footer.html');
?>
```

15. Save the file as `password.php`, place it in your Web directory, and test it in your Web browser (**Figures 8.20** and **8.21**).

✔ Tips

- If you delete every record from a table using the command `TRUNCATE tablename`, `mysqli_affected_rows()` will return *0*, even if the query was successful and every row was removed. This is just a quirk.
- If an `UPDATE` query runs but does not actually change the value of any column (for example, a password is replaced with the same password), `mysqli_affected_rows()` will return *0*.
- The `mysqli_affected_rows()` conditional used here could (and maybe should) also be applied to the `register.php` script to confirm that one record was added. That would be a more exacting condition to check than `if ($r)`.

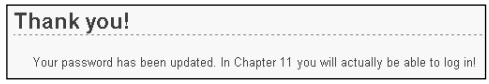


Figure 8.20 The password was changed in the database.

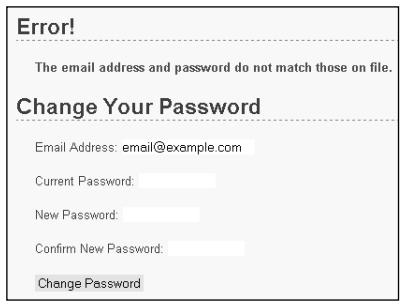


Figure 8.21 If the entered email address and password don't match those on file, the password will not be updated.