CSC 2720
Building Web Applications

Accessing MySQL from PHP
Connecting to MySQL

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>// Define named constants</td>
<td>DEFINE('DB_USER', 'username');</td>
</tr>
<tr>
<td>2</td>
<td>DEFINE('DB_PASSWORD', 'password');</td>
<td>DEFINE('DB_HOST', 'localhost');</td>
</tr>
<tr>
<td>3</td>
<td>DEFINE('DB_NAME', 'database_name');</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>// Establish connection</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>$dbc = @mysqli_connect ( DB_HOST, DB_USER,</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>DB_PASSWORD, DB_NAME )</td>
<td>OR exit('Could not connect to MySQL: ' .</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>mysqli_connect_error());</td>
</tr>
</tbody>
</table>

- Line 8: `mysqli_connect()` connects to a MySQL server running at DB_HOST and selects the database DB_NAME. Upon success, $dbc holds a reference to the database which is needed by subsequent mysqli_* function calls.
Connecting to MySQL

- Line 8: @ is an error suppression operator. It prevents the PHP error (when `mysqli_connect()` encounters an error) from being produced in the output.
  - The user don't need to see the error message.

- Line 10: "expr1 OR expr2" means "If expr1 is true, don’t evaluate expr2).
  - This kind of evaluation is called short circuit evaluation.
  - So at line 10, `exit()` will only be called if `mysqli_connect()` fails.

- Line 10: `exit($message)` outputs the value of `$message` and terminates.

- Line 11: `mysqli_connect_error()` returns a string containing a detailed error message about the failed connection.
Line 2: `mysqli_query()` sends a query to a server. It returns TRUE (or a `result object` for SELECT, SHOW, DESCRIBE, and EXPLAIN commands) on success or FALSE on failure.
Executing Queries

- Line 8: `mysqli_error()` returns a string containing a detailed error message about the failed query operation.

- Line 12: `mysqli_close()` closes the database connection.
  - If you don't close a database connection explicitly, the system will close it automatically when the script terminates. By closing a database connection explicitly as soon as you finished using it, the database can use the freed connection and resource to serve other requests.
$fn = "John";
$ln = "Doe";
$email = "johndoe@hotmail.com";
$pwd = "1234567";

$q = "INSERT INTO users " .
    "(first_name, last_name, email, pass, reg_date)" .
    "VALUES " .
    "('{$fn}', '{$ln}', '{$email}', SHA1('{$pwd}'), NOW())";

$result = @mysqli_query($dbc, $q);

if (!$result) {
    $err[] = "Fail to add user: " . mysqli_error($dbc);
}
mysqli_close($dbc);
Example: Adding a record

- Line 1-4: In practice these values are usually obtained from a HTML form.

- Line 9: `SHA1()` is an MySQL function that performs a one-way encryption on its parameter and returns a 40-character long encrypted string.
  - It is safer to store a password as encrypted string than as plain text
  - To compare passwords, you will need to compare the encrypted version.

- Line 9: `NOW()` is a function that returns the current date and time of the MySQL server (not the Web server).
Retrieving Query Results

```php
$q = "SELECT * FROM users";
$result = @mysqli_query($dbc, $q);

while ($row = mysqli_fetch_array($result)) {
    // $row is both an associative array and a numeric array
    $fname = $row['first_name'];

    // Assuming the 3rd column corresponds to 'last_name'
    $lname = $row[2];

    ...
}

mysqli_free_result($result);
mysqli_close($dbc);
```
Retrieving Query Results

- Line 2: A "SELECT" query returns a result set that contains multiple rows. For such command, `mysqli_query()` returns a reference to the result set.

- We can only retrieve one row of results at a time.

- Line 4: `mysqli_fetch_array($result)` returns the current row as an array (which can be processed as an associative or a numeric array). The function returns `NULL` when no more row is available.

- Line 15: `mysqli_free_result($result)` frees the resource used by $result.
Retrieving Query Results

- Related functions:
  - `mysqli_fetch_row()` – returns the result set as a numeric array
  - `mysqli_fetch_assoc()` – returns the result set as an associative array
  - `mysqli_fetch_object()` – returns the result set as an object
  - `mysqli_data_seek()` -- adjusts the result pointer to an arbitrary row in the result set

- Use different variables to hold the results of different queries.
  ```php
  $result = @mysqli_query($dbc, $q1);
  while ($row = mysqli_fetch_row($result)) {
    // Should use a different variable to hold the result
    $result = @mysqli_query($dbc, $q2);
    ...
  }
  ```
Escaping Special Characters

- Before using an input string in an SQL query, you should make sure all the special characters in the string are properly escaped.

- `mysqli_real_escape_string($dbc, $data)` returns a string containing all the characters in $data but with all the special characters in $data properly escaped.
Counting Returned or Affected Records

- `mysqli_num_rows($result)`
  returns the total number of rows in a result set returned by a SELECT query.
  - You can use this function to help you figure out if a SELECT command retrieves any row before starting to fetch anything from the result set.

- `mysqli_affected_rows($dbc)`
  returns the total number of rows affected by an INSERT, UPDATE, or DELETE query.
  - You can use this function to check if your command achieve its objective. e.g., is a DELETE command deleting exactly one row?
Using Prepared Statements

- With a prepared query, the SQL syntax is sent to MySQL first for "compilation". Then the specific values are sent separately.

- Advantage of prepared statements are
  - Greater security
  - Potentially better performance
Using Prepared Statements

- Without prepared statement:
  ```php
  $q = "SELECT first_name, last_name FROM users " . 
    "WHERE last_name=$lname";
  $result = mysqli_query($q);
  ```

- As prepared statement:
  ```php
  $q = "SELECT first_name, last_name FROM users " . 
    "WHERE last_name=?";
  $stmt = mysqli_prepare($dbc, $q);
  mysqli_stmt_bind_param($stmt, 's', $lname);
  mysqli_stmt_execute($stmt);
  ```
Using Prepared Statements

$q = "SELECT first_name, last_name FROM users " .
    "WHERE last_name=?";

- ? serves as a placeholder in the query

$stmt = mysqli_prepare($dbc, $q);

- Requests the server to compile the query

mysqli_stmt_bind_param($stmt, 's', $lname);

- Substitute the value of $lname into the location of the placeholder. 's' indicates the value is a string. ('d' for decimal, 'i' for integer, 'b' for binary data.)

mysqli_stmt_execute($stmt);

- Execute the query
Example: Using Prepared Statements

```php
$q = 'INSERT INTO users ' .
    '(first_name, last_name, email, pass, reg_date)' .
    ' VALUES (?, ?, ?, SHA1(?), NOW())';

$stm = mysqli_prepare($dbc, $q);

// Assume $users is an array containing the data of
// several users
for ($i = 0; $i < count($users); $i++) {
    mysqli_stmt_bind_param($stmt, 'ssss',
        $users['fname'], $users['lname'], $users['email'],
        $users['password']);

    mysqli_stmt_execute($stmt);
}
...
References and Resources

- MySQL
  - [http://www.mysql.com/](http://www.mysql.com/)

- PHP Manual for MySQL Improved Extension