INTRODUCTION TO WEB PROGRAMMING II (CSC 212)

A LECTURE NOTE

PREPARED BY

MUHAMMAD S. ALI

16/04/2021

How to use PHP with a MySQL database

- To create an object from a class, you code the new keyword, followed by the name of the class, followed by a set of parentheses. Within the parentheses, you code the arguments that are required by the class, separating multiple arguments with commas.
- To create a PDO object that connects to a MySQL database, you use the PDO class with the three arguments shown above: DSN (Data Source Name), username, and password.
- The DSN for a MySQL connection specifies the host computer for the MySQL database and the name of the database. If the MySQL database is running on the same server as PHP, you can use the localhost keyword to specify the host computer.
- The PDO (PHP Data Objects) extension to PHP defines a consistent interface for accessing databases. Since PDO supports most popular databases, this lets you write PHP code that can be used for more than one type of database.
- PDO is included with PHP 5.1 and is available as a PECL extension for PHP 5.0. However, PDO doesn't work with earlier versions of PHP.

- The syntax for creating an object from any class

  ```
  new ClassName(arguments);
  ```

- The syntax for creating a database object from the PDO class

  ```
  new PDO($dsn, $username, $password);
  ```

- The syntax for a DSN (Data Source Name) for a MySQL database

  ```
  mysql:host=host_address;dbname=database_name
  ```

- How to connect to a MySQL database named mydb

  ```
  $dsn = 'mysql:host=localhost;dbname=mydb';
  $username = 'mydb_operator';
  $password = '5tr0ngcracker';

  $db = new PDO($dsn, $username, $password); // creates PDO object
  ```

How to handle exceptions

- An **exception** is an object that contains information about an error that has occurred. Some PHP statements throw exceptions when they encounter an error. If an exception isn't handled, the application ends prematurely.
- To handle exceptions, you use a **try/catch** statement. First, you code a try block around any PHP statements that might throw an exception. Then, you code a catch block that catches the exception. This is known as ***exception handling***.
- The Exception class includes all types of exceptions, and all other exceptions are subclasses of the Exception class. The **PDOException** class is used for errors thrown by the PDO library.
- To call a method from any object, you code the name of the object, followed by ->, followed by the name of the method, followed by a set of parentheses. Within the parentheses, you code the arguments that are required by the method, separating multiple arguments with commas.
- All Exception objects provide a **getMessage** method that lets you get the error message.
- If an exception is thrown in the try block, any remaining statements in the try block are skipped and the statements in the catch block are executed. If no exceptions are thrown in the try block, the catch block is skipped.

- The syntax for a try/catch statement

```
try {
        // statements that might throw an exception
}
catch (ExceptionClass $exception_name) {
     // statements that handle the exception
}
```

- The syntax for executing a method of any object

```
$objectName->methodName(argumentList)
```

- How to handle a PDO exception

```
try {
      $db = new PDO($dsn, $username, $password);
      echo '<p>You are connected to the database!</p>';
} catch (PDOException $e) {
      $error_message = $e->getMessage();
       echo "<p>An error occurred while connecting to the database: $error_message </p>";
}
```

- How to handle any type of exception

```
try {
        // statements that might throw an exception
} catch (Exception $e) {
        $error_message = $e->getMessage();
         echo "<p>Error message: $error_message </p>";
}
```

How to get and modify data

- Once you connect to a database, you can use the connection to execute **SELECT** statements that retrieve data and **INSERT**, **UPDATE**, and **DELETE** statements that modify data.
- To prepare a SQL statement, you use the **prepare** method of the PDO object. It requires just one argument, which is the **SQL** statement to be executed.
- To execute a SQL statement, you use the **execute** method of the **PDOStatement** object.
- To identify a parameter in a SQL statement, you can code a colon (**:**) followed by the name of the parameter.
- To bind a variable to a parameter, you can use the **bind** method of the PDOStatement object.
- If the SQL statement is a SELECT statement that returns a result set, the execute method stores the result set in the PDOStatement object. Then, you can get the data from the PDOStatement object by using the techniques in the next two slides.

How to execute SELECT statements

- A method of the PDO class for preparing a SQL statement

  prepare($sql_statement) - Prepares the specified SQL statement for execution and returns a PDOStatement object. Within the SQL statement, you can use a colon (:) to add parameters to the statement.

- Two methods of the PDOStatement class for executing a statement

  bindValue($param, $value) - Binds the specified value to the specified parameter in the prepared statement.

  execute() - Executes the prepared statement.

- How to execute a SQL statement that doesn't have parameters

```
$query = "SELECT * FROM products";

$statement = $db->prepare($query);

$statement->execute();
```

- How to execute a SQL statement that has a parameter

```
$query = "SELECT * FROM products
            WHERE categoryID = :category_id";
$statement = $db->prepare($query);
$statement->bindValue(':category_id', $category_id);
$statement->execute();
```

How to work with the first row of a result set

- If a SELECT statement returns a result set that contains just *one row*, you can get data from that row by using the **fetch** method of the PDOStatement object as shown in the slide below. This method returns an array for the next row in the result set.
- An array can store one or more elements.
- To refer to the elements in an array, you use an index. If you use numeric indexes, 0 refers to the first element, 1 to the second element, and so on.
- You can use the fetch method of a PDOStatement object to get an array for the first row (or next row) of a result set. Then, you can use column names or numeric indexes to access the data that's stored in that row
- Two more methods of the PDOStatement class

  fetch() - Returns an array for the next row in the result set. This array is indexed by both a string index for the column name and a numeric index for the column position. If no array is available, this method returns a FALSE value.

  closeCursor() - Closes the cursor and frees the connection to the server so other SQL statements may be issued.

- Code that returns a result set that contains one row

```
$query = 'SELECT productCode, productName, listPrice
FROM products
WHERE productID = :product_id';
$statement = $db->prepare($query);
$statement->bindValue(':product_id', $product_id);
$statement->execute();
$product = $statement->fetch();
$statement->closeCursor();
```

- Code that uses a string index to access each column

```
$product_code = $product['productCode'];
$product_name = $product['productName'];
$product_list_price = $product['listPrice'];
```

- Code that uses a numeric index to access each column

```
$product_code = $product[0];
$product_name = $product[1];
$product_list_price = $product[2];
```

How to work with all the rows of a result set

- To get the data from all of the rows of a result set, you can use the **foreach** statement as shown in the slide below.
- When a PDOStatement object contains a result set, you can use the **fetchAll** method to return an array that contains that result set. Then, you can use the **closeCursor** method to close the connection to the database.
- To loop through the elements in an array, you can use a foreach statement to define a foreach loop that processes one element of the array each time through the loop.
- The second syntax for the foreach statement makes it easier to use the foreach statement with other control statements because it doesn't require the use of braces.
- When you use the second syntax with other control statements, you code a colon after the clause that starts each control statement, and you end each control statement with an appropriate end statement. For example, you end an if statement with an **endif** statement.
- Another method of the PDOStatement class

  fetchAll() - Returns an array for all of the rows in the result set.

- A query method that returns a result set of two or more rows

```
$query = 'SELECT productCode, productName, listPrice
FROM products WHERE categoryID = :category_id;'
$statement = $db->prepare($query);
$statement->bindValue(":category_id", $category_id);
$statement->execute();
$products = $statement->fetchAll();
$statement->closeCursor();
```

- How to use a foreach statement to display the result set in an HTML table

```
<?php foreach ($products as $product) { ?>
    <tr>
        <td><?php echo $product['productCode']; ?></td>
        <td><?php echo $product['productName']; ?></td>
        <td><?php echo $product['listPrice']; ?></td>
    </tr>
<?php } ?>
```

- Another syntax for the foreach statement that works better within PHP tags

```
<?php foreach ($products as $product) : ?>
    <tr>
        <td><?php echo $product['productCode']; ?></td>
        <td><?php echo $product['productName']; ?></td>
        <td><?php echo $product['listPrice']; ?></td>
    </tr>
<?php endforeach; ?>
```

How to execute INSERT statements

- How to execute an INSERT statement

```
$category_id = 1;
$code = 'strat';
$name = 'Fender Stratocaster';
$price = 699.99;
$query = "INSERT INTO products
            (categoryID, productCode, productName, listPrice)
          VALUES
            (:category_id, ':code', ':name', :price)";
$statement = $db->prepare($query);
$statement->bindValue(':category_id', $category_id);
$statement->bindValue(':code', $code);
$statement->bindValue(':name', $name);
$statement->bindValue(':price', $price);
$statement->execute();
$statement->closeCursor();
```

How to execute UPDATE statement

- How to execute an UPDATE statement

```
$product_id = 4; $price = 599.99;
$query = "UPDATE products
        SET listPrice = :price
        WHERE productID = :product_id";
$statement = $db->prepare($query);
$statement->bindValue(':price', $price);
$statement->bindValue(':product_id', $product_id);
$statement->execute();
$statement->closeCursor();
```

How to execute DELETE statements

- How to execute a DELETE statement

```
$product_id = 4;
$query = "DELETE FROM products
          WHERE productID = :product_id";
$statement = $db->prepare($query);
$statement->bindValue(':product_id', $product_id);
$statement->execute();
$statement->closeCursor();
```

How to use the MVC pattern

- The MVC (Model-View-Controller) pattern is commonly used to structure web applications that have significant processing requirements. That makes them easier to code and maintain.
- The model consists of the PHP files that represent the data of the application.
- The view consists of the HTML and PHP files that represent the user interface of the application.
- The controller consists of the PHP files that receive requests from users, get the appropriate data from the model, and return the appropriate views to the users.
- Before you can write code that uses the MVC pattern, you need to learn how to code your own (**custom**) **functions**.
- When you create a function, you can code a ***parameter*** *list* within parentheses. A parameter list contains one or more parameters.
- To *return data* from a function, you code a ***return*** statement in the body of the function. This statement *ends the execution* of the function and returns the specified value.
- A variable that's declared outside of a function isn't available within the function. This is called a ***variable's scope***. You can use the ***global*** keyword to make the variable available in the function, later you will learn a better way to work with variable scope.

- How to code a function
  - The syntax function

```
function function_name([parameter_list]) {
        // statements that are executed by the function
}
```

- A function with one parameter that returns an array of products

```
function get_products_by_category($category_id) {
        global $db;
        $query = 'SELECT * FROM products
                        WHERE products.categoryID = :category_id
                        ORDER BY productID';
        $statement = $db->prepare($query);
        $statement->bindValue(":category_id", $category_id);
        $statement->execute();
        $products = $statement->fetchAll();
        $statement->closeCursor();
        return $products;
}
```

- A function with four parameters that adds a product

```
function add_product($category_id, $code, $name, $price) {
    global $db;
    $query = 'INSERT INTO products
                (categoryID, productCode, productName, listPrice)
              VALUES
                (:category_id, :code, :name, :price)';
    $statement = $db->prepare($query);
    $statement->bindValue(':category_id', $category_id);
    $statement->bindValue(':code', $code);
    $statement->bindValue(':name', $name);
    $statement->bindValue(':price', $price);
    $statement->execute();
    $statement->closeCursor();
}
```

- How to call a function A function call with one argument and a returned array

  ```
  $products = get_products_by_category($category_id);
  ```

- A function call with four arguments and no returned value

  ```
  add_product($category_id, $code, $name, $price);
  ```

- How to redirect requests
  - In lesson 1, you learned how to use the **include** function to forward a request from one PHP file to another. When you forward a request, all processing takes place on the server.
  - You can use the **header** function to redirect a request to another URL. When you redirect a request, you return a response to the browser that has a **Location header** that specifies a page. This header causes the browser to make a new request for the specified URL.
  - The *header* function and the *include* function affect the browser's address bar differently. When you use the include function, the browser will display the URL of the original page. When you use the header function, the browser will display the URL of the page you redirected to.
- A built-in function that you can use to redirect a request

  header($header) - Sends an HTTP header to the browser. For example, you can use this function to send an HTTP Location header to the browser to redirect the browser to another URL.

- The header function

```
header('Location: .');
header('Location: ..');
header('Location: ./profile');
header('Location: error.php');
header('Location: http://www.jsiit.edu.ng');
```

- How to redirect a request

```
if ($action == 'delete_product') {
        $product_id = filter_input(INPUT_POST, 'product_id', FILTER_VALIDATE_INT);
        if ($product_id != NULL || $product_id != FALSE) {
                delete_product($product_id);
                header("Location: .");
        }
}
```

- How to redirect a request that includes a parameter

```
if ($action == 'delete_product') {

    $product_id = filter_input(INPUT_POST, 'product_id',FILTER_VALIDATE_INT);
    $category_id = filter_input(INPUT_POST, 'category_id',FILTER_VALIDATE_INT);

    if ($category_id != NULL || $category_id != FALSE ||$product_id != NULL ||
    $product_id != FALSE) {

        delete_product($product_id);

        header("Location: .?category_id=$category_id");

        }

}
```