Web Design and Development II

Week 11: MySQLi Basics

1.1 PHP MySQL Basics Overview

i. Create Database & Table
ii. Connection
iii. inserting
iv. Selection
v. Updating
vi. Deletion

1.2 PHP Connect to MySQL

PHP 5 and later can work with a MySQL database using:

- **MySQLi extension** (the "i" stands for improved)
- **PDO (PHP Data Objects)**

Earlier versions of PHP used the MySQL extension. However, this extension was deprecated in 2012.

1.3 MySQLi Connection

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

1.4 Closing the Connection

The connection will be closed automatically when the script ends. To close the connection before, use the following:

Example (MySQLi Object-Oriented)

- `$conn->close();`

Example (MySQLi Procedural)

- `mysqli_close($conn);`

1.5 Creating a Table

```
CREATE TABLE Lecturers (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP
)
```

1.6 MYSQL Database Constraints

Constraints are the rules enforced on the data columns of a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the **database**. **Constraints** could be either on a column level or a table level.

i. **NOT NULL** - Each row must contain a value for that column, null values are not allowed.

ii. **DEFAULT value** - Set a default value that is added when no other value is passed.

iii. **UNSIGNED** - Used for number types, limits the stored data to positive numbers and zero.

iv. **AUTO INCREMENT** - MySQL automatically increases the value of the field by 1 each time a new record is added.

v. **PRIMARY KEY** - Used to uniquely identify the rows in a table.

vi. **UNIQUE KEY-**

- The column with PRIMARY KEY setting is often an ID number and is often used with AUTO_INCREMENT

2.0 Inserting Data Into MySQL Table

The INSERT INTO statement is used to add new records to a MySQL table:

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

2.1 Timestamp and auto_increement

If a column is ***AUTO_INCREMENT*** (like the "id" column) or ***TIMESTAMP*** (like the "reg_date" column), it is no need to be specified in the SQL query; MySQL will automatically add the value.

2.2 Insertion Code

```
$sql = "INSERT INTO Lecturers(firstname, lastname, email)
VALUES ('Onesmus', 'Kitili', 'onessykei@gmail.com')";
if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}
$conn->close();
?>
```

```
$sql = "INSERT INTO Lecturers (firstname, lastname, email)
    VALUES ('Kennedy', 'Hadullo', 'khadullo@gmail.com')";
// use exec() because no results are returned
    $conn->exec($sql);
    echo "New record created successfully";
```

2.3 MSQLi Insertion

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

$sql = "INSERT INTO Lecturers (firstname, lastname, email)
VALUES ('Kennedy', 'Hadullo', 'khadullo@gmail.com')";

if ($conn->query($sql) === TRUE) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . $conn->error;
}

$conn->close();
?>
```

2.4 Prepared Statements & Bound Parameters

In **database management systems**, a prepared statement or parameterized statement is a feature used to execute the same or similar database statements repeatedly with high efficiency. Typically used with **SQL statements** such as queries or updates, the prepared statement takes the form of a template into which certain constant values are substituted during each execution.

2.5 How prepared statements work

1. **Prepare**: An SQL statement template is created and sent to the database. Certain values are left unspecified, called parameters (labeled "?"). Example: INSERT INTO MyGuests VALUES(?, ?, ?)

2. **Parse:** The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it.
3. **Execute:** At a later time, the application binds the values to the parameters, and the database executes the statement.

The application may execute the statement as many times as it wants with different values

Example PS

```
// prepare and bind
$stmt = $conn->prepare("INSERT INTO Lecturers (firstname, lastname,
email) VALUES (?, ?, ?)");
$stmt->bind_param("sss", $firstname, $lastname, $email);

// set parameters and execute
$firstname = "Kitili";
$lastname = "Onesmus";
$email = "onessykei@gmail.com";
$stmt->execute();

echo "New records created successfully";

$stmt->close();
$conn->close();
?>
```

Explanation

- "INSERT INTO Lecturers (firstname, lastname, email) VALUES (?, ?, ?)"
- In our SQL, we insert a question mark (?) where we want to substitute in an integer, string, double, or blob value.
    - Then, have a look at the bind_param() function:
    - $stmt->bind_param("sss", $firstname, $lastname, $email);
- This function binds the parameters to the SQL query and tells the database what the parameters are.
- The "sss" argument lists the types of data that the parameters are.
- The s character tells MySQL that the parameter is a string.
- The argument may be one of four types.

i - integer

d - double

s - string

b - BLOB

- We must have one of these for each parameter.
- By telling MySQL what type of data to expect, we minimize the risk of SQL injections.

3.0 Select Data With MySQLi

```
$sql = "SELECT id, firstname, lastname FROM Lecturers";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
   // output data of each row


   while($row = $result->fetch_assoc()) {
      echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " .
$row["lastname"]. "<br>";
   }
} else {
   echo "0 results";
}
$conn->close();
?>
```

First, we set up an SQL query that selects the ***id, firstname, and lastname*** columns from the Lecturers table. The next line of code runs the query and puts the resulting data into a variable called **$result.** Then, the function **num_rows()** checks if there are more than zero rows returned. If there are more than zero rows returned, the function **fetch_assoc()** puts all the results into an associative array that we can loop through. The **while() loop** loops through the result set and output the data from the id, firstname, and lastname columns.

4.0 Delete Data From Table Using MySQLi and PDO

The DELETE statement is used to delete records from a table:

```
DELETE FROM table_name
WHERE some_column = some_value
```

Notice the WHERE clause in the DELETE syntax:

The WHERE clause specifies which record or records that should be deleted. If you omit the WHERE clause, all records will be deleted!

The following examples delete the record with id=3 in the "Lecturers" table

```
// SQL to delete a record
$sql = "DELETE FROM Lecturers WHERE id=3";

if ($conn->query($sql) === TRUE) {
    echo "Record deleted successfully";
} else {
    echo "Error deleting record: " . $conn->error;
}

$conn->close();
?>
```

5.0 Updating Using MySQLi and PDO

The UPDATE statement is used to update existing records in a table:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Notice the WHERE clause in the UPDATE syntax:

The WHERE clause specifies which record or records that should be updated. If you omit the WHERE clause, all records will be updated!

```
$sql = "UPDATE Lecturers SET lastname='Doe' WHERE id=2";

    // Prepare statement
    $stmt = $conn->prepare($sql);

    // execute the query
    $stmt->execute();

    // echo a message to say the UPDATE succeeded
    echo $stmt->rowCount() . " records UPDATED successfully";
    }
catch(PDOException $e)
    {
    echo $sql . "<br>" . $e->getMessage();
    }

$conn = null;
?>
```