Experiment No. 5

Title: Database and PHP integration using PDO

Batch: A3 **Roll No.: 16010421073** **Experiment No:5**

Aim: Write PHP program for database activities using PDO.

Resources needed: Windows OS, Web Browser, Editor, XAMPP Server

---

Pre Lab/ Prior Concepts:
Students should have prior knowledge of mysql database and PHP constructs.

Theory:
The PHP Data Objects (PDO) extension defines a lightweight, consistent interface for accessing databases in PHP. Each database driver that implements the PDO interface can expose database-specific features as regular extension functions. Note that you cannot perform any database functions using the PDO extension by itself; you must use a database-specific PDO driver to access a database server.
PDO provides a data-access abstraction layer, which means that, regardless of which database you're using, you use the same functions to issue queries and fetch data. PDO does not provide a database abstraction; it doesn't rewrite SQL or emulate missing features. You should use a full-blown abstraction layer if you need that facility.

Installation/Configuration Requirement
When installing PDO as a shared module, the php.ini file needs to be updated so that the PDO extension will be loaded automatically when PHP runs. You will also need to enable any database specific drivers there too; make sure that they are listed after the pdo.so line, as PDO must be initialized before the database-specific extensions can be loaded.

```
extension=php_pdo.dll
extension=php_pdo_firebird.dll
extension=php_pdo_informix.dll
extension=php_pdo_mssql.dll
extension=php_pdo_mysql.dll
extension=php_pdo_oci.dll
extension=php_pdo_oci8.dll
extension=php_pdo_odbc.dll
extension=php_pdo_pgsql.dll
extension=php_pdo_sqlite.dll
```

Connecting to mysql

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
?>
```

If there are any connection errors, a PDOException object will be thrown. You may catch the exception if you want to handle the error condition, or you may opt to leave it for an application global exception handler that you set up via set_exception_handler().

```
<?php
```

```
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
    foreach($dbh->query('SELECT * from FOO') as $row) {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Error!: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Closing the connection

Upon successful connection to the database, an instance of the PDO class is returned to your script. The connection remains active for the lifetime of that PDO object. To close the connection, you need to destroy the object by ensuring that all remaining references to it are deleted—you do this by assigning null to the variable that holds the object. If you don't do this explicitly, PHP will automatically close the connection when your script ends.

For eg.,

```
$dbh = null; should be added at the end of the file
```

Persistent connection

Many web applications will benefit from making persistent connections to database servers. Persistent connections are not closed at the end of the script, but are cached and re-used when another script requests a connection using the same credentials. The persistent connection cache allows you to avoid the overhead of establishing a new connection every time a script needs to talk to a database, resulting in a faster web application.

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass, array(
    PDO::ATTR_PERSISTENT => true
));
?>
```

If you wish to use persistent connections, you must set PDO::ATTR_PERSISTENT in the array of driver options passed to the PDO constructor. If setting this attribute with PDO::setAttribute() after instantiation of the object, the driver will not use persistent connections.

Transactions are typically implemented by "saving-up" your batch of changes to be applied all at once; this has the nice side effect of drastically improving the efficiency of those updates. In other words, transactions can make your scripts faster and potentially more robust (you still need to use them correctly to reap that benefit).

TRANSACTIONS AND AUTOCOMMIT

Unfortunately, not every database supports transactions, so PDO needs to run in what is known as "auto-commit" mode when you first open the connection. Auto-commit mode means that every query that you run has its own implicit transaction, if the database supports it, or no transaction if the

database doesn't support transactions. If you need a transaction, you must use the PDO::beginTransaction() method to initiate one. If the underlying driver does not support transactions, a PDOException will be thrown (regardless of your error handling settings: this is always a serious error condition). Once you are in a transaction, you may use PDO::commit() or PDO::rollBack() to finish it, depending on the success of the code you run during the transaction.

Example: In the following sample, let's assume that we are creating a set of entries for a new employee, who has been assigned an ID number of 23. In addition to entering the basic data for that person, we also need to record their salary. It's pretty simple to make two separate updates, but by enclosing them within the PDO::beginTransaction() and PDO::commit() calls, we are guaranteeing that no one else will be able to see those changes until they are complete. If something goes wrong, the catch block rolls back all changes made since the transaction was started, and then prints out an error message.

```
<?php
try {
  $dbh = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2',
     array(PDO::ATTR_PERSISTENT => true));
  echo "Connected\n";
} catch (Exception $e) {
  die("Unable to connect: " . $e->getMessage());
}

try {
  $dbh->setAttribute(PDO::ATTR_ERRMODE,  PDO::ERRMODE_EXCEPTION);

  $dbh->beginTransaction();
  $dbh->exec("insert into staff (id, first, last) values (23, 'Joe', 'Bloggs')");
  $dbh->exec("insert into salarychange (id, amount, changedate)
     values (23, 50000, NOW())");
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

PREPARED STATEMENTS

Prepared Statement can be thought of as a kind of compiled template for the SQL that an application wants to run, that can be customized using variable parameters.

Prepared statements offer two major benefits:

The query only needs to be parsed (or prepared) once, but can be executed multiple times with the same or different parameters. When the query is prepared, the database will analyze, compile and optimize its plan for executing the query. For complex queries this process can take up enough time that it will noticeably slow down an application if there is a need to repeat the same query many

times with different parameters. By using a prepared statement the application avoids repeating the analyze/compile/optimize cycle. This means that prepared statements use fewer resources and thus run faster.
The parameters to prepared statements don't need to be quoted; the driver automatically handles this. If an application exclusively uses prepared statements, the developer can be sure that no SQL injection will occur (however, if other portions of the query are being built up with unescaped input, SQL injection is still possible).
Prepared statements are so useful that they are the only feature that PDO will emulate for drivers that don't support them. This ensures that an application will be able to use the same data access paradigm regardless of the capabilities of the database.

Insert using Prepared Statement

```
<?php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (?, ?)");
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $value);

// insert one row
$name = 'one';
$value = 1;
$stmt->execute();

// insert another row with different values
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

Retrieval using Prepared statement

```
<?php
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");
$stmt->execute([$_GET['name']]);
foreach ($stmt as $row) {
print_r($row);
}
?>
```

Stored Procedure prepared statement

```
<?php
$stmt = $dbh->prepare("CALL sp_takes_string_returns_string(?)");
$value = 'hello';
$stmt->bindParam(1, $value, PDO::PARAM_STR|PDO::PARAM_INPUT_OUTPUT, 4000);

// call the stored procedure
$stmt->execute();
print "procedure returned $value\n";
?>
```

PDO has following error handling strategies

PDO::ERRMODE_SILENT

PDO will simply set the error code for you to inspect using the PDO::errorCode() and PDO::errorInfo() methods on both the statement and database objects;

PDO::ERRMODE_WARNING

In addition to setting the error code, PDO will emit a traditional E_WARNING message. This setting is useful during debugging/testing, if you just want to see what problems occurred without interrupting the flow of the application.

PDO::ERRMODE_EXCEPTION

In addition to setting the error code, PDO will throw a PDOException and set its properties to reflect the error code and error information. This setting is also useful during debugging, as it will effectively "blow up" the script at the point of the error, very quickly pointing a finger at potential problem areas in your code (remember: transactions are automatically rolled back if the exception causes the script to terminate).

Exception mode is also useful because you can structure your error handling more clearly than with traditional PHP-style warnings, and with less code/nesting than by running in silent mode and explicitly checking the return value of each database call.

Example #1 Create a PDO instance and set the error mode

```
<?php
$dsn = 'mysql:dbname=testdb;host=127.0.0.1';
$user = 'dbuser';
$password = 'dbpass';

$dbh = new PDO($dsn, $user, $password);
$dbh->setAttribute(PDO::ATTR_ERRMODE,  PDO::ERRMODE_EXCEPTION);

// This will cause PDO to throw a PDOException (when the table doesn't exist)
$dbh->query("SELECT wrongcolumn FROM wrongtable");
```

The above example will output:

Fatal error: Uncaught PDOException: SQLSTATE[42S02]: Base table or view not found: 1146 Table 'testdb.wrongtable' doesn't exist in /tmp/pdo_test.php:10
Stack trace:
#0 /tmp/pdo_test.php(10): PDO->query('SELECT wrongcol...')
#1 {main}
thrown in /tmp/pdo_test.php on line 10

Example #2 Create a PDO instance and set the error mode from the constructor

```
<?php
$dsn = 'mysql:dbname=test;host=127.0.0.1';
$user = 'googleguy';
$password = 'googleguy';

$dbh = new PDO($dsn, $user, $password, array(PDO::ATTR_ERRMODE => PDO::ERRMODE_WARNING));

// This will cause PDO to throw an error of level E_WARNING instead of an exception (when the table doesn't exist)
$dbh->query("SELECT wrongcolumn FROM wrongtable");
```

Activity: Write PHP program to work with PDO database.
Complete the following functionalities

- Creating a database.
- Creating a table.
- Inserting values into the table (values to be extracted from user input) using prepared statement.
- Retreiving values from the table using prepared statement and displaying to the user.
- Using Exception handling to show error wherever possible.
- Update some values of the table based on user input.

Output(Code with result Snapshot)

1) Github Link

Index.php

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="a.css">
</head>
<body>
  <div class="container">
    <div class="block1">
       <button onclick="displayPara(1)">MySql</button>
        <button onclick="displayPara(2)">MariaDb</button>
        <button onclick="displayPara(3)">Sql Lite</button>
        <button onclick="displayPara(4)">PgSQL</button>
        <button onclick="displayPara(5)">Firebase</button>
    </div>

    <div class="block2" id="paraContainer">
    </div>
    <div class="block3">
```

```
      <div class="form-container">
        <div>
          <h2>Insert Data</h2>
          <form method="POST">
            <label for="name">Name:</label>
            <input type="text" name="name" required>
            <label for="roll_no">Roll No:</label>
            <input type="text" name="roll_no" required>
            <label for="marks">Marks:</label>
            <input type="text" name="marks" required>
            <input type="submit" name="insert" value="Insert">
          </form>
        </div>

        <div>
          <h2>Update Data</h2>
          <form method="POST">
            <label for="roll_no">Roll No to Update:</label>
            <input type="text" name="roll_no" required>
            <label for="new_name">New Name:</label>
            <input type="text" name="new_name" required>
            <label for="new_roll_no">New Roll No:</label>
            <input type="text" name="new_roll_no" required>
            <label for="new_marks">New Marks:</label>
            <input type="text" name="new_marks" required>
            <input type="submit" name="update" value="Update">
          </form>
        </div>

        <div>
          <h2>Delete Data</h2>
          <form method="POST">
            <label for="delete_roll_no">Roll No to Delete:</label>
            <input type="text" name="delete_roll_no" required>
            <input type="submit" name="delete" value="Delete">
          </form>
        </div>
      </div>

  </div>
  <div class="block4">
     <table>
        <tr>
          <th>Name</th>
          <th>Roll No</th>
          <th>Marks</th>
        </tr>
        <?php
```

```
class TableRows extends RecursiveIteratorIterator {
  function __construct($it) {
    parent::__construct($it, self::LEAVES_ONLY);
  }
  function current() {
    return "<td>" . parent::current(). "</td>";
  }
  function beginChildren() {
    echo "<tr>";
  }
  function endChildren() {
    echo "</tr>\n";
  }
}

$servername = "localhost";
$username = "root";
$password = "";
$dbname = "result";

try {
  $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
  $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // INSERT data
  if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["insert"])) {
    $name = $_POST["name"];
    $rollNo = $_POST["roll_no"];
    $marks = $_POST["marks"];

    $stmt = $conn->prepare("INSERT INTO student (name, roll_no, marks)
VALUES (:name, :roll_no, :marks)");
    $stmt->bindParam(':name', $name);
    $stmt->bindParam(':roll_no', $rollNo);
    $stmt->bindParam(':marks', $marks);
    $stmt->execute();
  }

  // UPDATE data
  if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["update"])) {
    $rollNo = $_POST["roll_no"];
    $newName = $_POST["new_name"];
    $newRollNo = $_POST["new_roll_no"];
    $newMarks = $_POST["new_marks"];

    $stmt = $conn->prepare("UPDATE student SET name = :new_name, roll_no =
:new_roll_no, marks = :new_marks WHERE roll_no = :roll_no");
```

```
              $stmt->bindParam(':roll_no', $rollNo);
              $stmt->bindParam(':new_name', $newName);
              $stmt->bindParam(':new_roll_no', $newRollNo);
              $stmt->bindParam(':new_marks', $newMarks);
              $stmt->execute();
            }

            // DELETE data
            if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["delete"])) {
              $deleteRollNo = $_POST["delete_roll_no"];

              $stmt = $conn->prepare("DELETE FROM student WHERE roll_no =
:delete_roll_no");
              $stmt->bindParam(':delete_roll_no', $deleteRollNo);
              $stmt->execute();
            }

            // SELECT data
            $stmt = $conn->prepare("SELECT * FROM student");
            $stmt->execute();

            $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
            foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as
$k=>$v) {
              echo $v;
            }
          } catch(PDOException $e) {
            echo "Error: " . $e->getMessage();
          }

          $conn = null;
          ?>
        </table>
    </div>
  </div>
  <script src="index.js"></script>
</body>
</html>
```

a.css

```
  .container {
    display: grid;
```

```
    grid-template-columns: 1fr 1fr;
    grid-template-rows: 1fr 1fr;
  }
  .block1, .block2, .block3, .block4 {
    border: 2px solid black;
    padding: 20px;
  }
  table {
    width: 100%;
    border: solid 2px black;
  }
  th, td {
    border: 2px solid black;
    padding: 5px;
  }
  h2 {
    margin-top: 20px;
  }
  form {
    display: block;
    margin-bottom: 20px;
  }
  label, input {
    display: block;
    margin-bottom: 10px;
  }
```

index.js

```
function displayPara(num) {
    const paraContainer = document.getElementById("paraContainer");
    let paraText = "";

    switch (num) {
      case 1:
        paraText = `<pre><code>try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // INSERT data
    if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["insert"])) {
      $name = $_POST["name"];
      $rollNo = $_POST["roll_no"];
      $marks = $_POST["marks"];
```

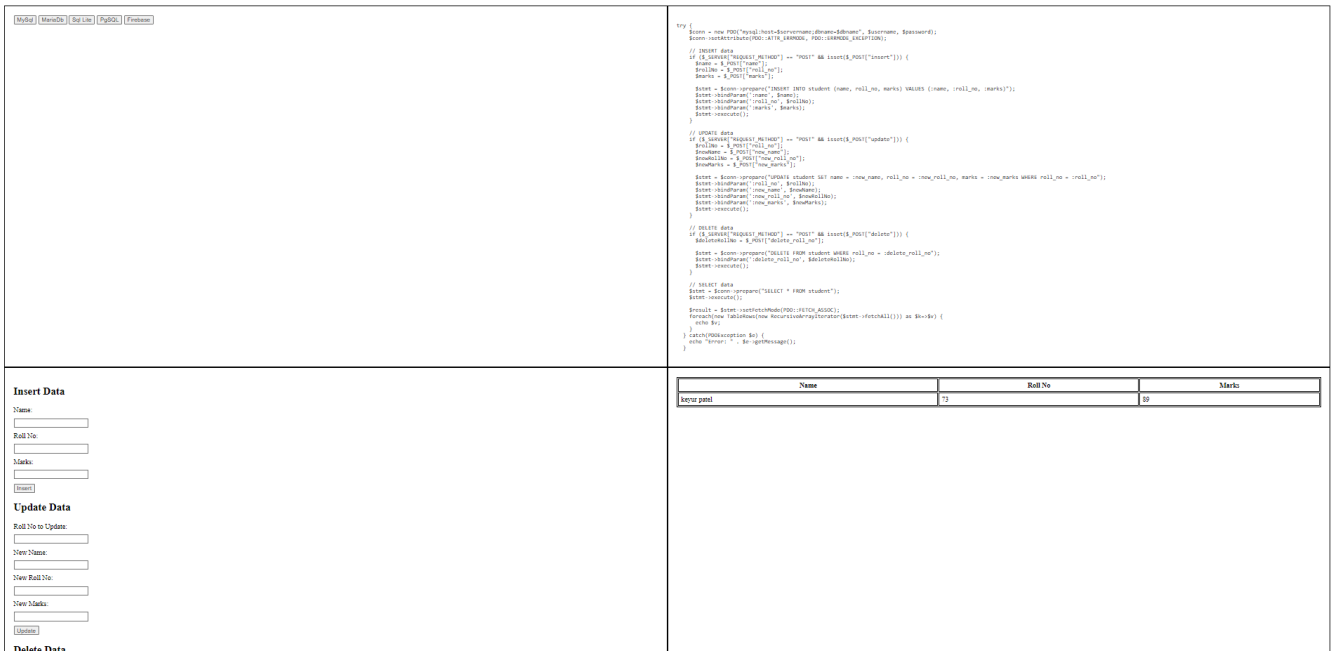
```
      $stmt = $conn->prepare("INSERT INTO student (name, roll_no, marks) VALUES
(:name, :roll_no, :marks)");
      $stmt->bindParam(':name', $name);
      $stmt->bindParam(':roll_no', $rollNo);
      $stmt->bindParam(':marks', $marks);
      $stmt->execute();
    }

    // UPDATE data
    if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["update"])) {
      $rollNo = $_POST["roll_no"];
      $newName = $_POST["new_name"];
      $newRollNo = $_POST["new_roll_no"];
      $newMarks = $_POST["new_marks"];

      $stmt = $conn->prepare("UPDATE student SET name = :new_name, roll_no =
:new_roll_no, marks = :new_marks WHERE roll_no = :roll_no");
      $stmt->bindParam(':roll_no', $rollNo);
      $stmt->bindParam(':new_name', $newName);
      $stmt->bindParam(':new_roll_no', $newRollNo);
      $stmt->bindParam(':new_marks', $newMarks);
      $stmt->execute();
    }

    // DELETE data
    if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["delete"])) {
      $deleteRollNo = $_POST["delete_roll_no"];

      $stmt = $conn->prepare("DELETE FROM student WHERE roll_no = :delete_roll_no");
      $stmt->bindParam(':delete_roll_no', $deleteRollNo);
      $stmt->execute();
    }

    // SELECT data
    $stmt = $conn->prepare("SELECT * FROM student");
    $stmt->execute();

    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
      echo $v;
    }
  } catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
  }</code></pre>`;
        break;
      case 2:
        paraText = `<pre><code>
```

```
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database_name";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
```

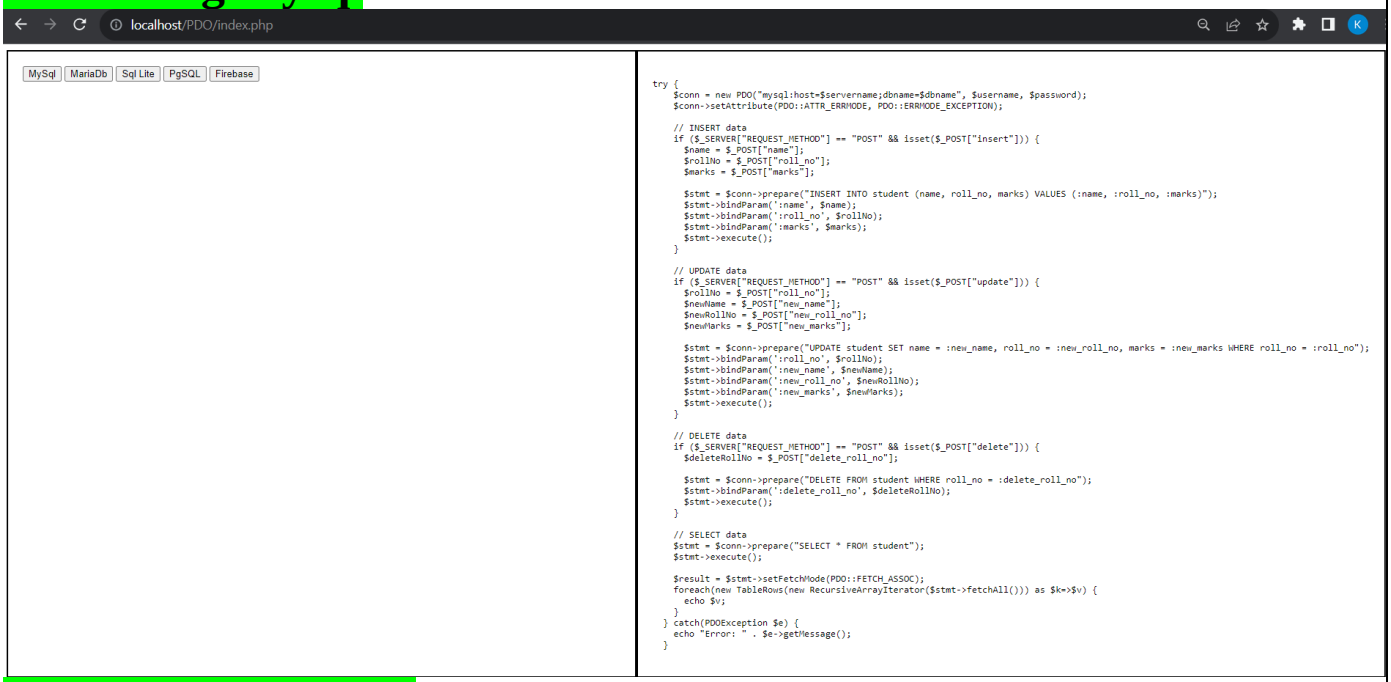
```
      </code></pre>`;
      break;
    case 3:
      paraText = `<pre><code>
      $dbname = "result";

try {
    $conn = new PDO("sqlite:$dbname");
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}


      </code></pre>`;
      break;
    case 4:
      paraText = `<pre><code>
      $host = "localhost";
$dbname = "result";
$username = "your_username_here";
$password = "your_password_here";

try {
    $conn = new PDO("pgsql:host=$host;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
```

```
  }
        </code></pre>`;
        break;
      case 5:
        paraText = `<pre><code>
        require __DIR__.'/vendor/autoload.php';

  use Kreait\Firebase\Factory;
  $factory = (new Factory())-> withProjectId('database-a818f')
  ->withServiceAccount('c:/xampp/htdocs/Keyur/MultipleDb/database-a818f-firebase-
adminsdk-xjo10-8d9bf0776b.json')
  ->withDatabaseUri('https://database-a818f-default-rtdb.firebaseio.com/');


  $database = $factory->createDatabase();

        </code></pre>`;
        break;
      default:
        paraText = "Invalid button.";
    }

    paraContainer.innerHTML = `<p>${paraText}</p>`;
  }
```

Screenshots

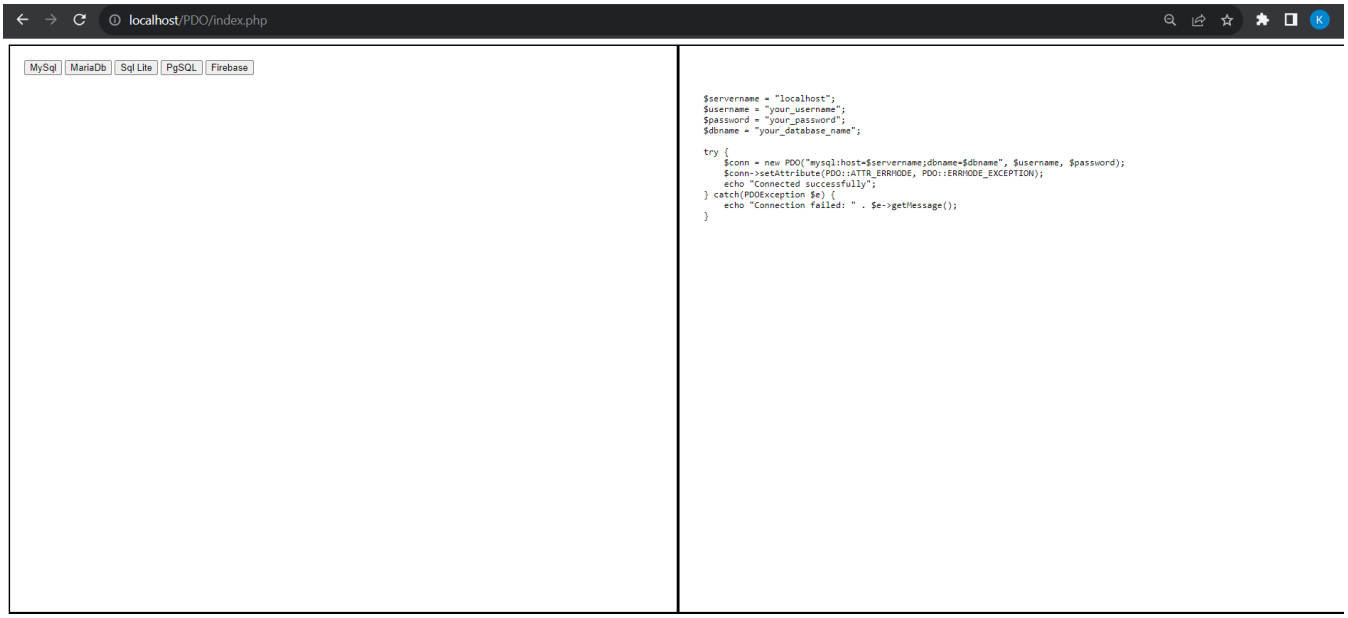


Onclicking Mysql

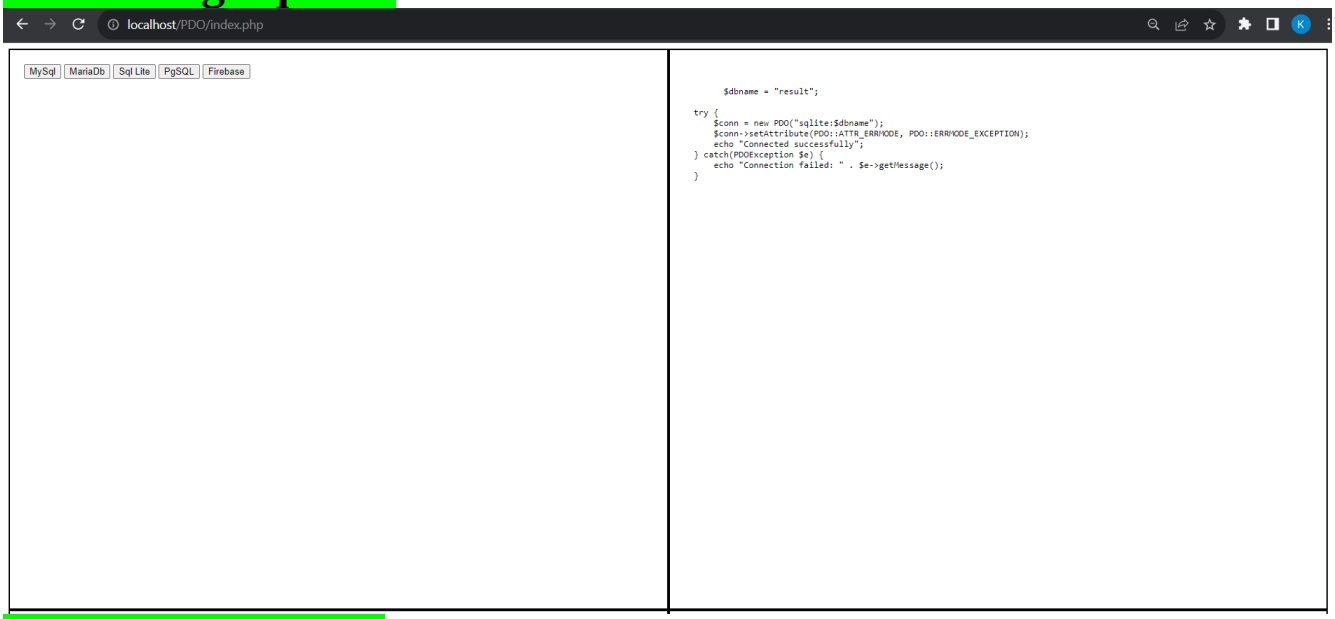



```php
try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    // INSERT data
    if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["insert"])) {
      $name = $_POST["name"];
      $rollNo = $_POST["roll_no"];
      $marks = $_POST["marks"];

      $stmt = $conn->prepare("INSERT INTO student (name, roll_no, marks) VALUES (:name, :roll_no, :marks)");
      $stmt->bindParam(':name', $name);
      $stmt->bindParam(':roll_no', $rollNo);
      $stmt->bindParam(':marks', $marks);
      $stmt->execute();
    }

    // UPDATE data
    if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["update"])) {
      $rollNo = $_POST["roll_no"];
      $newName = $_POST["new_name"];
      $newRollNo = $_POST["new_roll_no"];
      $newMarks = $_POST["new_marks"];

      $stmt = $conn->prepare("UPDATE student SET name = :new_name, roll_no = :new_roll_no, marks = :new_marks WHERE roll_no = :roll_no");
      $stmt->bindParam(':roll_no', $rollNo);
      $stmt->bindParam(':new_name', $newName);
      $stmt->bindParam(':new_roll_no', $newRollNo);
      $stmt->bindParam(':new_marks', $newMarks);
      $stmt->execute();
    }

    // DELETE data
    if ($_SERVER["REQUEST_METHOD"] == "POST" && isset($_POST["delete"])) {
      $deleteRollNo = $_POST["delete_roll_no"];

      $stmt = $conn->prepare("DELETE FROM student WHERE roll_no = :delete_roll_no");
      $stmt->bindParam(':delete_roll_no', $deleteRollNo);
      $stmt->execute();
    }

    // SELECT data
    $stmt = $conn->prepare("SELECT * FROM student");
    $stmt->execute();

    $result = $stmt->setFetchMode(PDO::FETCH_ASSOC);
    foreach(new TableRows(new RecursiveArrayIterator($stmt->fetchAll())) as $k=>$v) {
      echo $v;
    }
  } catch(PDOException $e) {
    echo "Error: " . $e->getMessage();
  }
```

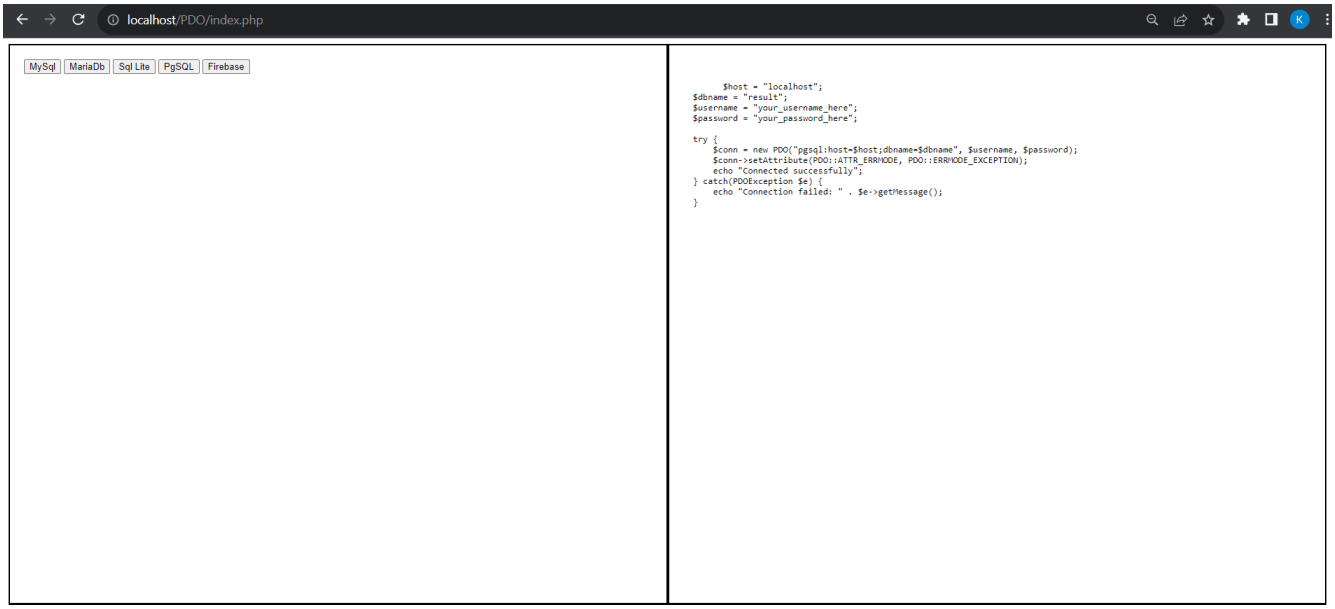
Onclicking Mariadb



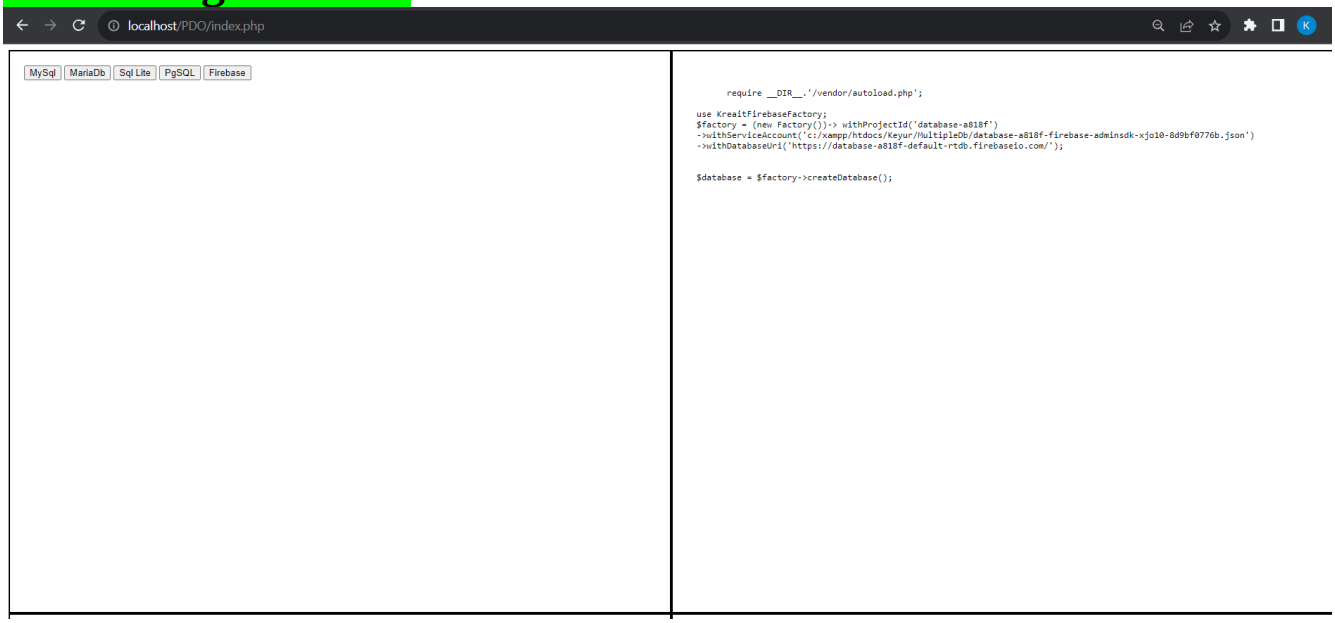
Onclicking Sql Lite



Onclicking PgSQL



Onclicking Firebase



Insert , Delete and Update (Name,Roll No and marks)

Insert Data

Name:

Roll No:

Marks:

Insert

Update Data

Roll No to Update:

New Name:

New Roll No:

New Marks:

Update

Delete Data

Roll No to Delete:

Delete

Name	Roll No	Marks

Insert(Added 3 names)

Insert Data

Name:

Roll No:

Marks:

Insert

Update Data

Roll No to Update:

New Name:

New Roll No:

New Marks:

Update

Delete Data

Roll No to Delete:

Delete

Name	Roll No	Marks
keyur patel	73	89
Aditya Ratho	87	88
Kushal	67	79

Update (Updated roll no 87)

Insert Data

Name:

Roll No:

Marks:

Insert

Update Data

Roll No to Update:

New Name:

New Roll No:

New Marks:

Update

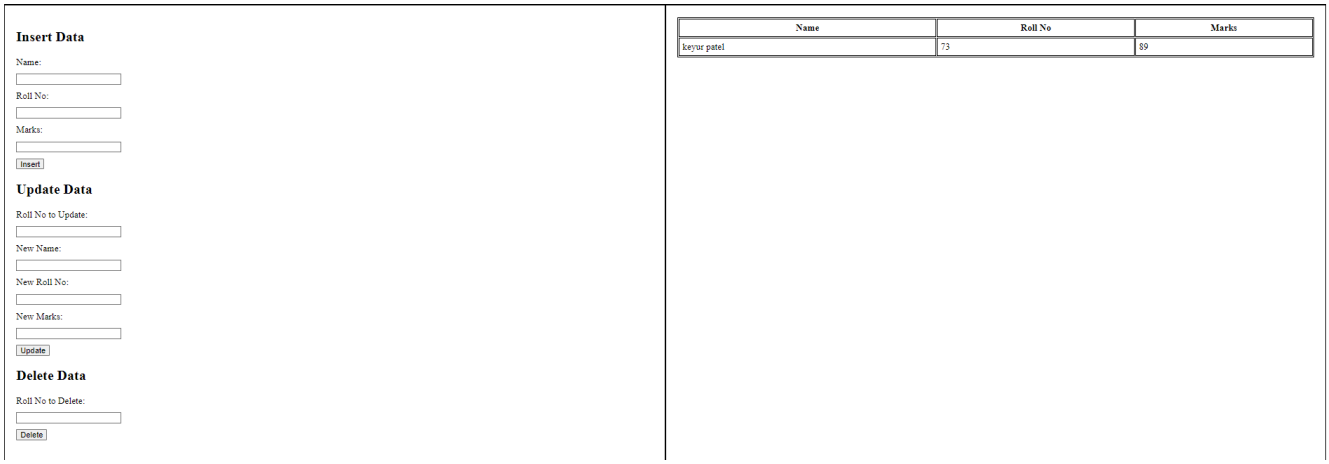
Delete Data

Roll No to Delete:

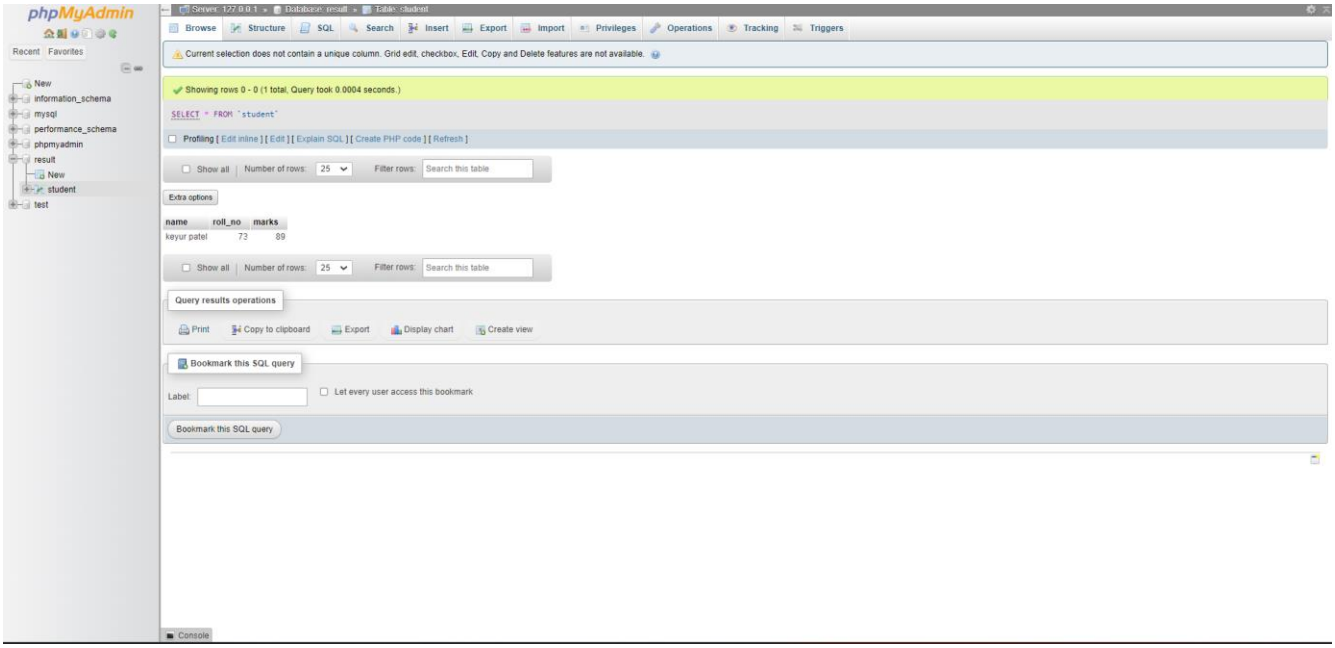
Delete

Name	Roll No	Marks
keyur patel	73	89
Mahesh	88	99
Kushal	67	79

Delete(removed two roll no 67 and 87)



On phpmyadmin



Code Pgsql

```php
<?php
$host = "localhost";
$dbname = "result";
$username = "your_username_here";
$password = "your_password_here";

try {
```

```
    $conn = new PDO("pgsql:host=$host;dbname=$dbname", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

Code Sqlite

```
<?php
$dbname = "result";

try {
    $conn = new PDO("sqlite:$dbname");
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

Code for MariaDb

```
<?php
$servername = "localhost";
$username = "your_username";
$password = "your_password";
$dbname = "your_database_name";

try {
    $conn = new PDO("mysql:host=$servername;dbname=$dbname", $username,
$password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connected successfully";
} catch(PDOException $e) {
    echo "Connection failed: " . $e->getMessage();
}
?>
```

Post Lab Questions:-

1. **Write in detail the importance of using Prepared Statements.**

Answer: Prepared statements are a crucial concept in database management and application development, particularly when interacting with relational databases like MySQL, PostgreSQL, or Oracle. They provide a more secure, efficient, and maintainable way to execute SQL queries compared to dynamically constructed SQL statements. Following are importance of it:

1. **Security:** Prepared statements help prevent SQL injection attacks, which are one of the most common and dangerous security vulnerabilities in web applications. SQL injection occurs when an attacker injects malicious SQL code into user input fields, causing unintended and potentially harmful database operations
2. **Performance:** Prepared statements can improve query execution performance. When you create a prepared statement, the database server compiles the SQL query once and creates an execution plan. Subsequent executions of the prepared statement reuse this plan, resulting in faster query execution compared to dynamically constructed queries, where the database has to parse and optimize the SQL statement every time it's executed.
3. **Database Optimization:** Databases can optimize prepared statements more effectively because they know the query structure in advance. This leads to better query plan optimization, potentially reducing query execution times and server resource usage.
4. **Maintainability:** Prepared statements make code more maintainable. By separating SQL queries from user input and parameter values, you create cleaner, more modular code. It's easier to read, understand, and maintain because you don't have to worry about manually escaping or concatenating values into SQL strings.

2. **Write in detail about PDO constructor function.**

Answer: The PDO (PHP Data Objects) constructor function is a key component in PHP for working with databases. It is used to establish a database connection using the PDO extension, which provides a consistent and efficient way to interact with various database management systems (DBMS) such as MySQL, PostgreSQL, SQLite, and more.

The PDO constructor function is used to create a new PDO object, which represents a database connection. It typically takes three main parameters:

1. **DSN (Data Source Name):**The DSN is a string that specifies the database driver, host, and database name, among other connection details. The format of the DSN varies depending on the database system you are connecting to. For example, here's a DSN for connecting to a MySQL database:

$dsn = "mysql:host=localhost;dbname=mydatabase";

In this example, mysql is the database driver, localhost is the host where the database is running, and mydatabase is the name of the database.

1. **Username:**This parameter represents the username used to authenticate with the database server. It is typically associated with the privileges and permissions granted to the user.
2. **Password:**The password is the corresponding password for the provided username. It is used for authentication with the database server.

Here's an example of using the PDO constructor to create a database connection to a MySQL database:

```
try {
$dsn = "mysql:host=localhost;dbname=mydatabase";
$username = "myuser";
$password = "mypassword";
// Create a new PDO instance
$pdo = new PDO($dsn, $username, $password);
// Set PDO attributes (optional)
```

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Now you have a valid database connection in $pdo
} catch (PDOException $e) {
// Handle any connection errors
echo "Connection failed: " . $e->getMessage();
        }
```

3. **Write a php program to return the id of the last inserted row.**

Answer: To return the ID of the last inserted row in a MySQL database using PHP, we can use the LAST_INSERT_ID() function provided by MySQL in combination with the PDO extension.

```
<?php
try {
// Create a new PDO connection to your MySQL database
$pdo = new PDO("mysql:host=localhost;dbname=your_database", "your_username", "your_password");
// Set the PDO error mode to exception
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
// Your SQL INSERT query
$sql = "INSERT INTO your_table (column1, column2) VALUES (:value1, :value2)";
// Prepare the SQL statement
$stmt = $pdo->prepare($sql);
// Bind parameters
$value1 = "Some Value";
$value2 = "Another Value";
$stmt->bindParam(':value1', $value1);
$stmt->bindParam(':value2', $value2);
// Execute the statement
$stmt->execute();
// Get the ID of the last inserted row
$lastInsertedID = $pdo->lastInsertId();
echo "Last Inserted ID: " . $lastInsertedID;
} catch (PDOException $e) {
echo "Error: " . $e->getMessage();
}
        ?>
```

4. **What are the essential components of a PDO database connection string?**

Answer: A PDO (PHP Data Objects) database connection string, also known as a DSN (Data Source Name), contains essential components that are required to establish a connection to a database. These components can vary slightly depending on the specific database management system (DBMS) you are connecting to, but in general, a PDO DSN typically consists of the following key elements:

1) **Database Driver or DSN Prefix:** This is the first part of the connection string and specifies the type of database you are connecting to. It's also referred to as the DSN prefix. Common database drivers include: MySQL: mysql

- This is the first part of the connection string and specifies the type of database you are connecting to. It's also referred to as the DSN prefix.
  Common database drivers include:
- MySQL: mysql
- PostgreSQL: pgsql
- SQLite: sqlite

- SQL Server: sqlsrv
- Oracle: oci

The host parameter specifies the hostname or IP address of the database server. If your database server is running on the same machine as your PHP script, you can often use "localhost" or "127.0.0.1". If your database server is on a different machine, you would specify its address here.
Port is an optional parameter that specifies the port number on which the database server is listening. If omitted, the default port for the database system is used. For example, MySQL uses port 3306 by default.
This parameter specifies the name of the specific database or schema within the database server that you want to connect to. It's important to ensure that the database you intend to access exists on the server.
These parameters are used for authentication. The username is the database user account you want to use, and the password is the corresponding password for that user.
Depending on the database system and the PDO driver, you may have additional options you can specify in the DSN. These options can include settings like character set, SSL configuration, and others. The format and availability of these options can vary between DBMS and PDO drivers.

2) Host and Port (Optional):

- The host parameter specifies the hostname or IP address of the database server. If your database server is running on the same machine as your PHP script, you can often use "localhost" or "127.0.0.1". If your database server is on a different machine, you would specify its address here.
- Port is an optional parameter that specifies the port number on which the database server is listening. If omitted, the default port for the database system is used. For example, MySQL uses port 3306 by default.

3) Database Name (Schema): This parameter specifies the name of the specific database or schema within the database server that you want to connect to. It's important to ensure that the database you intend to access exists on the server.

4) Username and Password: These parameters are used for authentication. The username is the database user account you want to use, and the password is the corresponding password for that user.

5) Additional Options (Optional):

Here's a template for a MySQL DSN with placeholders for the components:
"mysql:host=localhost;port=3306;dbname=mydatabase"

5. **Explain the concept of database connection pooling and how it can be implemented with PDO.**

Answer: Database connection pooling is a technique used to efficiently manage and reuse database connections in a web application. It helps improve the performance and scalability of database interactions by reducing the overhead associated with opening and closing database connections for every database operation. The concept of database connection pooling can be applied to PDO in PHP using third-party libraries or by implementing custom solutions.

Here's an explanation of the concept and how it can be implemented with PDO:

Concept of Database Connection Pooling:

1. **Connection Creation Overhead:** Establishing a new database connection is a relatively resource-intensive process. It involves network communication, authentication, and potentially other setup tasks. When a web application handles multiple database queries simultaneously, opening and closing connections for each query can lead to high overhead.
2. **Connection Pool:** Database connection pooling involves creating a pool of database connections at the start of the application or when needed. These connections are kept open and reused for multiple database queries.
3. **Connection Reuse:** When a PHP script needs to interact with the database, it borrows a connection from the pool, executes the query, and returns the connection to the pool. This avoids the overhead of creating and destroying connections for every query.

Implementation with PDO:

To implement database connection pooling with PDO in PHP, you can follow these steps:

Choose a Connection Pooling Library: There are third-party libraries available that provide connection pooling functionality for PDO. Popular options include: PDO Connection Pooling Libraries: Some libraries like spomky-labs/pdo-pool and illuminate/database (used in Laravel) offer built-in connection pooling for PDO. You can integrate them into your project.

Initialize the Connection Pool: Initialize the connection pool with a set of pre-established PDO connections when your application starts. The pool can be a shared resource accessible to different parts of your application.

Acquire and Release Connections: Whenever your PHP scripts need to interact with the database, they request a connection from the pool. After executing the query, they release the connection back to the pool for reuse.

Outcomes: CO3: Carry out database operations using PHP.

Conclusion: (Conclusion to be based on the objectives and outcomes achieved)

Hence, I understood the concept of Database and PHP integration using PDO. I also implemented CRUD operations by using PDO and prepared statements

Signature of faculty in-charge with date

References:

1. Thomson PHP and MySQL Web Development Addison-Wesley Professional , 5th Edition2016.
2. https://www.php.net/manual/en/book.pdo.php