MySQLi Object-Oriented

For checking the connection

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


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


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

Create database

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

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

  if ($conn->connect_error) {
      die("Connection failed: " . $conn->connect_error);
  }

  $sql = "CREATE DATABASE myDB";

  if ($conn->query($sql) === TRUE) {
      echo "Database created successfully";
  } else {
      echo "Error creating database: " . $conn->error;
  }

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

Create table

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

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

  // sql to create table
  $sql = "CREATE TABLE MyGuests (
  id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
  firstname VARCHAR(30) NOT NULL,
  lastname VARCHAR(30) NOT NULL,
  email VARCHAR(50),
  reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
  )";

  if ($conn->query($sql) === TRUE) {
      echo "Table MyGuests created successfully";
  } else {
      echo "Error creating table: " . $conn->error;
  }

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

PHP MySQL Insert Data

-
  ```
  <?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 MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com')";

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

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

Insert Multiple Records

-
  ```
  <?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 MyGuests (firstname, lastname, email)
  VALUES ('John', 'Doe', 'john@example.com');";
  $sql .= "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Mary', 'Moe', 'mary@example.com');";
  $sql .= "INSERT INTO MyGuests (firstname, lastname, email)
  VALUES ('Julie', 'Dooley', 'julie@example.com')";

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

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

(Vvp) PHP MySQL Prepared Statements

- Prepared Statements and Bound Parameters
- A prepared statement is a feature used to execute the same (or similar) SQL statements repeatedly with high efficiency. It is applicable with all the DML statements.

Prepared statements basically work like this:

- 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(?, ?, ?)
- The database parses, compiles, and performs query optimization on the SQL statement template, and stores the result without executing it
- 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
- Compared to executing SQL statements directly, prepared statements have three main advantages:
- Prepared statements reduce parsing time as the preparation on the query is done only once (although the statement is executed multiple times)
- Bound parameters(bind parameters) minimize bandwidth to the server as you need send only the parameters each time, and not the whole query
- Prepared statements are very useful against SQL injections, because parameter values, which are transmitted later using a different protocol, need not be correctly escaped. If the original statement template is not derived from external input, SQL injection cannot occur.

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

  $conn = new mysqli($servername, $username, $password, $dbname);

  if ($conn->connect_error) {
      die("Connection  failed:  " . $conn->connect_error);
  }
  // prepare and bind
  $stmt = $conn->prepare("INSERT INTO MyGuests (firstname, lastname, email)  VALUES (?, ?, ?)");
  $stmt->bind_param("sss",  $firstname, $lastname, $email);  //first parameter decides the type of data to be entered
  // set parameters and execute
  $firstname = "John";
  $lastname = "Doe";
  $email = "john@example.com";
  $stmt->execute();

  $firstname = "Mary";
  $lastname = "Moe";
  $email = "mary@example.com";
  $stmt->execute();

  $firstname = "Julie";
  $lastname = "Dooley";
  $email = "julie@example.com";
  $stmt->execute();

  echo "New records created successfully";

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

Select Data With MySQLi

```
<?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 = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);//execute the query being passed in the form of parameter.

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 MyGuests 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 outputs the data from the id, firstname and lastname columns.

Select and Filter Data With MySQLi

-
  ```
  <?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 = "SELECT id, firstname, lastname FROM MyGuests WHERE lastname= ‘Doe’ ";
  $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();
  ?>
  ```

Update Data

-
  ```
  <?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 = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

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

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

Delete data from the table

-
  ```
  <?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 = “delete from myguests where id=2"; //delete the record for the id=2

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

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

PDO connectivity –PHP Data Objects

- Connections can be established by creating instances of the PDO base class. It doesn't matter which driver you want to use; you always use the PDO class name.
- The constructor accepts parameters for specifying the database source (known as the DSN) and optionally for the username and password .
- Using PDO the connectivity to any database can be done.

PDO steps required

- A PDO database connection requires you to create a new PDO object with a Data Source Name (DSN), Username, and Password.

- **The DSN** defines the type of database, the name of the database, and any other information related to the database if required. These are the variables and values we stated inside the dbconfig.php file, referenced one time by the line require_once in the databaseconnect.php.

- In the latter, you will find the try…catch.. code. This means that the script will try to MySQL connect using the code provided, but if there is a problem, the code in the catch section will run. You can use the catch block to display connect error messages or run alternate code if the try block fails.

- If the connection is successful, it will print out the message "Connected to $dbname at $host successfully." However, if the attempt fails, the catch code will show a simple error message and kill the script.

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().

Handling connection errors

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

Points to note

- 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.

Closing a connection

-
  ```
  <?php  //connection is being set
  $dbh = new PDO('mysql:host=localh
  ost;dbname=test', $user, $pass);
  // use the connection here
  $sth = $dbh-
  >query('SELECT * FROM foo');

  // and now we're done; close it
  $sth = null;  //assign the
  variable to null
  $dbh = null;  //assign the
  connection to null
  ?>
  ```

PDO::setAttribute

- public PDO::setAttribute (int $attribute , mixed $value) : bool
- PDO::ATTR_CASE: Force column names to a specific case.
- PDO::CASE_LOWER: Force column names to lower case.
- PDO::CASE_NATURAL: Leave column names as returned by the database driver.
- PDO::CASE_UPPER: Force column names to upper case.

- PDO::ATTR_ERRMODE: Error reporting.

- PDO::ERRMODE_SILENT: Just set error codes.- This is the default mode. 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; if the error resulted from a call on a statement object, you would invoke the PDOStatement::errorCode() or PDOStatement::errorInfo() method on that object

- PDO::ERRMODE_WARNING: Raise E_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: Throw exceptions.-In addition to setting the error code, PDO will throw a PDOException and set its properties to reflect the error code and error information.

Persistent connections

- 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.
- Persistent connections were designed to have one-to-one mapping to regular connections. That means that you should always be able to replace persistent connections with non-persistent connections, and it won't change the way your script behaves. It may (and probably will) change the efficiency of the script, but not its behavior!

-
  ```
  <?php
  $dbh = new PDO('mysql:host=localhost;
  dbname=test', $user, $pass, array(
      PDO::ATTR_PERSISTENT => true
  ));
  ?>
  ```
- The value of the PDO::ATTR_PERSISTENT option is converted to bool (enable/disable persistent connections), unless it is a non-numeric string, in which case it allows to use multiple persistent connection pools. This is useful if different connections use incompatible settings, for instance, different values of PDO::MYSQL_ATTR_USE_BUFFERED_QUERY.

PHP PDO exec

- The PDO exec() executes an SQL statement and returns the number of affected rows.(no. of rows filtered after applying the query).
- <?php

- $dsn = "mysql:host=localhost;dbname=mydb";
- $user = "username";
- $passwd = “password";

- $pdo = new PDO($dsn, $user, $passwd);

- $id = 12;

- $nrows = $pdo->exec("DELETE FROM countries WHERE id IN (1, 2, 3)");

- echo "The statement affected $nrows rows\n"; ?>

- In this SQL statement, we delete rows with ids 1, 2, and 3. The number of deleted rows is stored in the $nrows variable.

PHP PDO fetch style

- The fetch style parameter controls how the next row will be returned to the caller. For instance, the PDO::FETCH_ASSOC returns an array indexed by column name, PDO::FETCH_NUM returns an array indexed by column number, and the PDO::FETCH_BOTH returns an array indexed by both column name and indexed column number. The default fetch style is PDO::FETCH_BOTH.

```
<?php

$dsn = "mysql:host=localhost;dbname=mydb";
$user = "user12";
$passwd = "12user";

$pdo = new PDO($dsn, $user, $passwd);

$stm = $pdo->query("SELECT * FROM countries");

$rows = $stm->fetchAll(PDO::FETCH_NUM);

foreach($rows as $row) {

    printf("$row[0] $row[1] $row[2]\n");
}
```