



“People Build Nations, We Build People”



## Introduction

For tomorrow’s lesson, we will cover the basics of MySQLi and PHP forms. MySQLi is a PHP extension used for interacting with MySQL databases, offering secure and efficient database operations. We'll learn how to connect to a database, run queries, and insert data using key MySQLi functions like `mysqli_connect()`, `mysqli_query()`, and `mysqli_close()`. We'll also explore how HTML forms work with PHP to capture user input, process it using `$_POST`, and store it in a MySQL database.

### 1. Introduction to MySQLi

**MySQLi (MySQL Improved)** is a PHP extension that allows you to interact with MySQL databases. It provides a more secure and feature-rich way to work with databases compared to the older `MySQL` extension. MySQLi supports both procedural and object-oriented programming.

- **Why Use MySQLi?**

- It allows you to perform secure database operations.
- It supports prepared statements for preventing SQL injection attacks.
- It provides functions for both connecting to and interacting with MySQL databases.

**A. Connecting to a Database :** Before you can interact with a MySQL database in PHP, you need to establish a connection to the MySQL server. The `mysqli_connect()` function is used to open this connection. Let’s break down how to use this function and its parameters.

```
mysqli_connect($servername, $username, $password, $dbname);
```

**B. Parameters of `mysqli_connect()` :**

- **`$servername`** (Required): This is the hostname or IP address of the MySQL server. In most cases, for local development, this is usually set to `localhost`. For remote servers, it will be the IP address or domain name of the server where MySQL is hosted. Example: `"localhost"`
- **`$username`** (Required): The MySQL username you want to authenticate with. This is typically set to `root` for local servers like XAMPP or WAMP, but it should be the username that has the proper permissions to access the database. Example: `"root"`
- **`$password`** (Required): The password associated with the MySQL username. For local servers, this may be empty by default, but it should be filled with the password if you are connecting to a remote server.
- **`$dbname`** (Required): This is the name of the specific database you want to connect to after the connection is established. Make sure that the database exists on the MySQL server.

## Example 1: Basic Connection

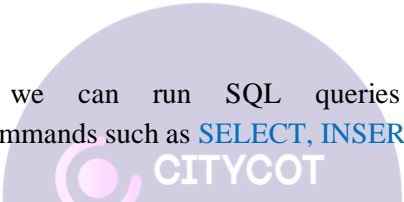
```
$con = mysqli_connect("localhost","my_user","my_password","my_db");  
if ($con) {  
    echo "Connection Success";  
}  
  
else {  
    echo "Connection Failed";  
}
```

## Error Handling for Connections

If the connection fails, you can use `mysqli_connection_error()` to get a detailed error message. This helps in debugging issues related to the database connection.

## C. Running SQL Queries

Once connected to the database, we can run SQL queries using the `mysqli_query()` function. This function is used for executing SQL commands such as [SELECT](#), [INSERT](#), [UPDATE](#) and [DELETE](#).



```
<?php  
$con = mysqli_connect("localhost", "root", "password", "mydb");  
$sql = "CREATE TABLE students (  
    ID INT PRIMARY KEY,  
    First_Name VARCHAR(255),  
    Last_Name VARCHAR(255),  
    Place_Of_Birth VARCHAR(255),  
    Country VARCHAR(255)  
)";  
  
if (mysqli_query($con, $sql)) {  
    echo "Table 'students' created successfully!";  
} else {  
    echo "Error creating table: " . mysqli_error($con);  
}  
mysqli_close($con);  
?>
```

## D. Retrieving Error Messages

The `mysqli_error()` function is used to retrieve a detailed error message from MySQL when a query fails. This is useful for debugging and understanding why a query may not be working as expected.

## E. Closing the Connection

After performing all the database operations, it's important to close the connection to the database using `mysqli_close()`. This releases resources and ensures efficient database management.

## Inserting Data into MySQL Using MySQLi

Inserting data into a MySQL database using MySQLi is a fundamental operation. You can use the INSERT INTO SQL statement to add data to the database.

- A. Insert Data into MySQL Table :To insert data, we use the `mysqli_query()` function with an SQL query. Below is the structure:

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

In PHP, it would look like this:

```
$sql = "INSERT INTO customders (name, email, phone, address)
VALUES ('Mohamed Ahed', 'iammohamed@gmail.com', '722', 'Bosaso');"
```

Here:

- customers is the table name.
- name, email, phone, and address are the columns of the table.
- The values for these columns are inserted in the VALUES section.

- B. Using `mysqli_query()` to Execute the Insert

To insert the data into the database, use the `mysqli_query()` function, which executes the SQL query. Here's the full process:

```
<?php
$sql = "INSERT INTO customers (name, email, phone, address)
      VALUES ('Mohamed Ahed', 'iammohamed@gmail.com', '722', 'Bosaso')";

if (mysqli_query(mysql: $con, query: $sql)) {
    echo "Record inserted successfully.";
} else {
    echo "Error: " . mysqli_error(mysql: $con);
}

mysqli_close(mysql: $con);
?>
```

## PHP MySQLi Exercise

1. **Create a Database:**
  - Create a new database called EAU.

## 2. Create a Table:

- Inside the EAU database, create a table called students with the following columns:
  - studentid (Primary Key, Auto Increment)
  - studentname (VARCHAR, 100 characters)
  - phone (VARCHAR, 15 characters)
  - email (VARCHAR, 100 characters)
  - faculty\_name (VARCHAR, 50 characters)

## 3. Insert Data into the Table:

- Insert the following data into the students table

## How HTML Forms Work with PHP

HTML forms are essential for capturing user input in web applications. When users submit a form, the data is sent to the server for processing. PHP can then access and manipulate this data based on the input provided. Let's explore how HTML forms interact with PHP to create dynamic applications.

### 1. Form Creation in HTML

The first step is to create the HTML form, which is where users input their data. Forms in HTML are created using the `<form>` tag, which can specify the action (where the data will be sent) and the method (how the data will be sent). Here is an example of a simple HTML form:

```
<form action="process.php" method="post">  
  
  Name: <input type="text" name="name"><br>  
  Email: <input type="email" name="email"><br>  
  <input type="submit" value="Submit">  
  
</form>
```

• **Action:** The `action` attribute specifies the PHP file (e.g., `process.php`) that will handle the form data after the user submits it.

• **Method:** The `method` attribute specifies how the data will be sent: `GET` or `POST`. In this example, we're using `POST`, which sends data in the HTTP request body, making it more secure for sensitive information.

## 2. Form Submission

When the user fills out the form and clicks the "Submit" button, the browser sends the data to the PHP script specified in the action attribute. The data is sent either via the **GET** or **POST** method, and the PHP script is ready to process it.

3. **Processing Form Data in PHP**: In the PHP file (e.g., process.php), the form data can be accessed using PHP's superglobal arrays: `$_GET`, `$_POST`, or `$_REQUEST`. These arrays hold the form data based on the method used, here is an example of processing the form data using **`$_POST`**

```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = $_POST['student_name'];
    $email = $_POST['email'];

    echo "Welcome, " . $name . "<br>";
    echo "Your email is: " . $email . "<br>";
}
?>
```

- **`$_POST`**: is used to retrieve data sent via the **POST** method. In this case, we're getting the values of the `name` and `email` fields from the form.

## Storing HTML Form Data in a MySQL Database Using MySQLi

In this section, we will learn how to store form data in a MySQL database using PHP and MySQLi. Storing user input from HTML forms in a database is a common task when creating dynamic web applications. Once the data is stored in the database, you can later retrieve it, display it on your site, or process it for other uses.

### ⇒ Prerequisites

Before we start, make sure you have the following:

- A MySQL database where you want to store the form data.
- A table in the database ready to store the data.

### ⇒ HTML Form to Collect User Data

The first step in creating a form is to write the HTML code. This form will collect the students' name, email, phone number, and message. When the user clicks "Submit," the data will be sent to a PHP file (**`submit.php`**) for processing.

Here's the HTML form:

```
<form action="submit.php" method="post">
  Name: <input type="text" name="studentname"><br>
  Email: <input type="email" name="email"><br>
  Phone: <input type="text" name="phone"><br>
  <input type="submit" value="Submit">
</form>
```

### ⇒ PHP Script to Insert Data into MySQL

Now, let's create the PHP script (**submit.php**) that will process the form data and insert it into the database.

```
<?php

$name = $_POST['name'];
$email = $_POST['email'];
$phone = $_POST['phone'];
$message = $_POST['message'];

$sql = "INSERT INTO users (name, email, phone, message)
      VALUES ('$name', '$email', '$phone', '$message')";

if (mysqli_query(mysql: $conn, query: $sql)) {
    echo "New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error(mysql: $conn);
}

mysqli_close(mysql: $conn);
?>
?>
```

### Explanation of the Code:

- **Fetching Form Data:** The form data sent via POST is captured using **`$_POST['field_name']`**. These values are assigned to variables like **`$name`, `$email`, `$phone`, and `$message`**.
- **SQL Query:** We prepare the INSERT INTO query to insert the form data into the users table. The placeholders (**`'$name'`, `'$email'`, etc.**) are replaced with the user input.
- **Executing the Query:** The **`mysqli_query()`** function is used to execute the SQL query. If successful, a success message is displayed; otherwise, the error message is shown.
- **Closing the Connection:** It's important to close the connection to the database using **`mysqli_close()`** to free up resources.

## Conclusion

To summarize, we studied the following MySQLi functions:

- **mysqli\_connect()** for connecting to the database.
- **mysqli\_query()** for executing SQL queries.
- **mysqli\_close()** for closing the database connection.
- **mysqli\_connect\_error()** for handling connection errors.

Also, we see how to **use PHP's \$\_POST** to handle form data and store it securely in a MySQL database, covering the entire process from form creation to data insertion.

