Tutorial6

Part A: SQL Crash Course

Objective: Before writing PHP code, we must understand the raw SQL commands used to communicate with the database. We will run these directly in the **phpMyAdmin's SQL tab**.

1. Setting the Stage (DDL)

These commands define the structure of your data.

Create Database

First, we need a container for our data.

```
CREATE DATABASE tech_store;
```

Select the Database

Specify the database you want to use with MySQL.

```
USE tech_store;
```

Create Products Table

- Add an `id` column as an `INT` that is `AUTO_INCREMENT` and the `PRIMARY KEY`.
- Add a `name` column as `VARCHAR(100)` and set it to `NOT NULL`.
- Add a `price` column as `DECIMAL(10, 2)` for currency values.
- Add a `category` column as `VARCHAR(50)` for the product type.

2. Manipulating Data (CRUD Operations)

These are the core operations: **C**reate, **R**ead, **U**pdate, and **D**elete.

INSERT (Create)

Add data to the table.

Insert a single row

```
INSERT INTO products (name, price, category)
VALUES ('iPhone 15', 999.00, 'Smartphone');
```

Write a SQL Query to insert multiple rows at once

SELECT (Read)

Retrieve data from the table.

See everything

```
SELECT  FROM products;
```

See specific columns

```
SELECT name, price FROM products;
```

Filter results (WHERE clause)

```
SELECT  FROM products WHERE category = 'Smartphone';
```

Sort results

```
SELECT  FROM products ORDER BY price DESC;
```

UPDATE

Modify existing data.

> Warning: Always use a WHERE clause, or you will update every row!

Change the price of the product with ID 1

```
UPDATE products
SET price = 1099.00
WHERE id = 1;
```

DELETE

Remove data.

Delete a specific product

```
DELETE FROM products WHERE id = 3;
```

3. Advanced Management

ALTER TABLE

Used to add or remove columns in an existing table.

Add a new column for 'Stock Quantity'

```
ALTER TABLE products ADD COLUMN stock_quantity INT DEFAULT 100;
```

DROP

Deletes the entire table or database (Dangerous!)

Do not run this unless you want to start over by creating the products table

```
DROP TABLE products;
```

Part B: PHP Integration

Topic 1: Database Design & Normalization

In **Part A**, we stored "Category" as text. In a real app, this is redundant. Imagine storing “Laptop” as a category for 100s of rows.

We use **Normalization** to split data into related tables.

The Plan:

1. **Categories Table:** Holds unique categories.
2. **Products Table:** Holds products and links to Categories using a **Foreign Key**.

Run this in PHPMyAdmin to reset your table structure for the PHP application.

```
DROP TABLE IF EXISTS products;
```

Create Parent Table

```
CREATE TABLE categories (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

Create Child Table

```
CREATE TABLE products (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL,
    stock_quantity INT DEFAULT 0,
    category_id INT,
    FOREIGN KEY (category_id) REFERENCES categories(id)
);
```

Add some default categories

```
INSERT INTO categories (name)
VALUES
('Laptops'),
('Phones'),
```

`('Accessories');`

Topic 2: Connecting PHP to MySQL (PDO)

We use **PDO** (PHP Data Objects) because it is secure and supports multiple database types.

Let’s create a file named `db.php`.

Task: Write code to set up the connection with the following attributes.

```
$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Without it (Default), if your SQL fails, PHP often continues running the script silently.

You might see a blank page or broken data without knowing why.

With it (`ERRMODE_EXCEPTION`), PHP "crashes" explicitly with a detailed error message telling you exactly what went wrong (e.g., "Table 'products' doesn't exist").

```
$pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
```

Without this line (Default), PHP returns duplicate data—once with the column name and once with a number index.

`['name' => 'iPhone', 0 => 'iPhone']`

With this line (`FETCH_ASSOC`), PHP returns just the clean column names.

`['name' => 'iPhone']`

Topic 3: Create (Insert Data)

> For Security, we use Prepared Statements (using ? placeholders) to prevent SQL Injection attacks.

Create `create.php`.

```
<?php

 require 'db.php';

 try {
    $stmt = $pdo->query("SELECT * FROM categories");
    $categories = $stmt->fetchAll();
} catch (PDOException $e) {
    die("SQL Error: " . $e->getMessage());
}


 if($_SERVER['REQUEST_METHOD']==="POST"){
    $name = $_POST['name'];
    $price = $_POST['price'];
    $stock = $_POST['stock'];
    $category_id = $_POST['category_id'];

    $sql = "INSERT INTO products (name, price, stock_quantity, category_id)
VALUES(?,?,?,?)";

    $stmt = $pdo->prepare($sql);

    $execute = $stmt->execute([$name,$price, $stock, $category_id]);

    if($execute){

        echo "<h3 style='color:blue;'>Product Successfully Added! Check dat
```

```
abase.</h3>";
    }

    }

?>

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Create Product</title>
</head>
<body>
<form method="POST">
    Name: <input type="text" name="name" required> <br><br>
    Price: <input type="number" name="price" step="0.01" required><br><
br>
    Stock: <input type="number" name="stock" required><br><br>

    Category:
    <select name="category_id">
        <?php
            foreach($categories as $category):
        ?>
            <option value="<?=htmlspecialchars($category['id'])?>"><?=htmls
pecialchars($category['name'])?></option>
        <?php endforeach;?>
    </select><br><br>
    <button type="submit">Add Product</button>
</form>
</body>
</html>
```

Task: Use the form to add a Laptop and a Phone to your database.

Topic 4: Read (Select with JOIN)

To display the Category Name (e.g., "Laptops") instead of the ID number (e.g., "1"), we perform a `LEFT JOIN`.

Hints:

- **Setup:** Create `index.php` and include your database connection at the top using `require 'db.php';`.
- **Query:** Write and execute a SQL query using `$pdo->query()` that uses a `LEFT JOIN` to fetch all product data along with their matching category names.
- **Table Structure:** Create standard HTML table tags with a header row (`<th>`) for Name, Category, Price, Stock, and Actions.
- **Loop:** Start a PHP `while` loop that iterates through the results using `$row = $stmt->fetch()`.
- **Display Data:** Inside the loop, output a table row (`<tr>`) containing cells (`<td>`) that display the name, category, price, and stock variables.
- **Add Links:** In the final cell, create "Edit" and "Delete" anchor tags (`<a>`) that append the current row's ID to the URL (e.g., `edit.php?id=<?= $row['id'] ?>`).

Task: Sort the items by Price (Highest first).

Topic 5: Update (Edit Data)

Updating requires two steps:

1. **GET:** Fetch existing data to fill the form.
2. **POST:** Submit changes to the database.

Create `edit.php`.

```
<?php
require 'db.php';

$id = $_GET['id'] ?? null;
if (!$id) die("ID required");


$stmt = $pdo->prepare("SELECT * FROM products WHERE id = ?");
$stmt->execute([$id]);
$product = $stmt->fetch();


if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $name = $_POST['name'];
    $price = $_POST['price'];

    $sql = "UPDATE products SET name = ?, price = ? WHERE id = ?";
    $stmt = $pdo->prepare($sql);
    $stmt->execute([$name, $price, $id]);

    header("Location: index.php");
    exit;
}
?>

<form method="post">
    Name: <input type="text" name="name" value="<?= htmlspecialchars
```

```
($product['name']) ?>"><br>
    Price: <input type="number" step="0.01" name="price" value="<?= $pr
oduct['price'] ?>"><br>
    <button type="submit">Update</button>
</form>
```

Task: The `edit.php` above is missing the **Stock** field. Add the stock input and update the SQL query to save stock changes.

Topic 6: Delete

We use the ID from the URL to delete a specific row.

Create `delete.php`.

Hints:

- **Setup:** Create `delete.php` and include your database connection at the top using `require 'db.php';`.
- **Validation:** Use an `if` statement with `isset($_GET['id'])` to ensure a specific product ID was passed in the URL.
- **Preparation:** Inside the `if` block, prepare a secure SQL statement using `$pdo->prepare("DELETE FROM products WHERE id = ?")`
- **Execution:** Run the deletion by calling `$stmt->execute([$id])`, passing the ID captured from the URL into the placeholder
- **Redirection:** Use `header("Location: index.php");` to immediately send the user back to the inventory list after the deletion
- **Cleanup:** Add `exit;` immediately after the header function to ensure the script stops executing.