EXP 7: Do design using Object Oriented approach and hence highlight Cohesion and Coupling in the given design.

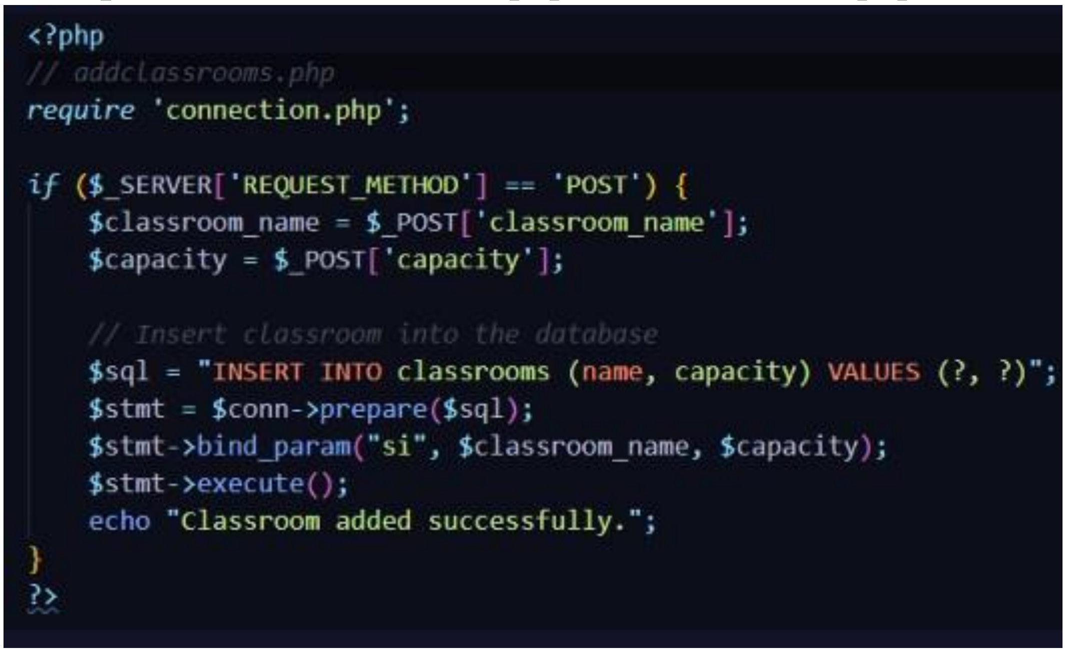
Course Scheduling System

Cohesion:

In the Course Scheduler, we designed our modules to exhibit high cohesion by carefully defining each module's responsibility. This clear separation allows each module to focus on a specific part of the scheduling process, making the codebase more modular, easier to debug, and straightforward to extend.

1. FUNCTIONAL COHESION

Example Files: addclassrooms.php, deleteclassroom.php

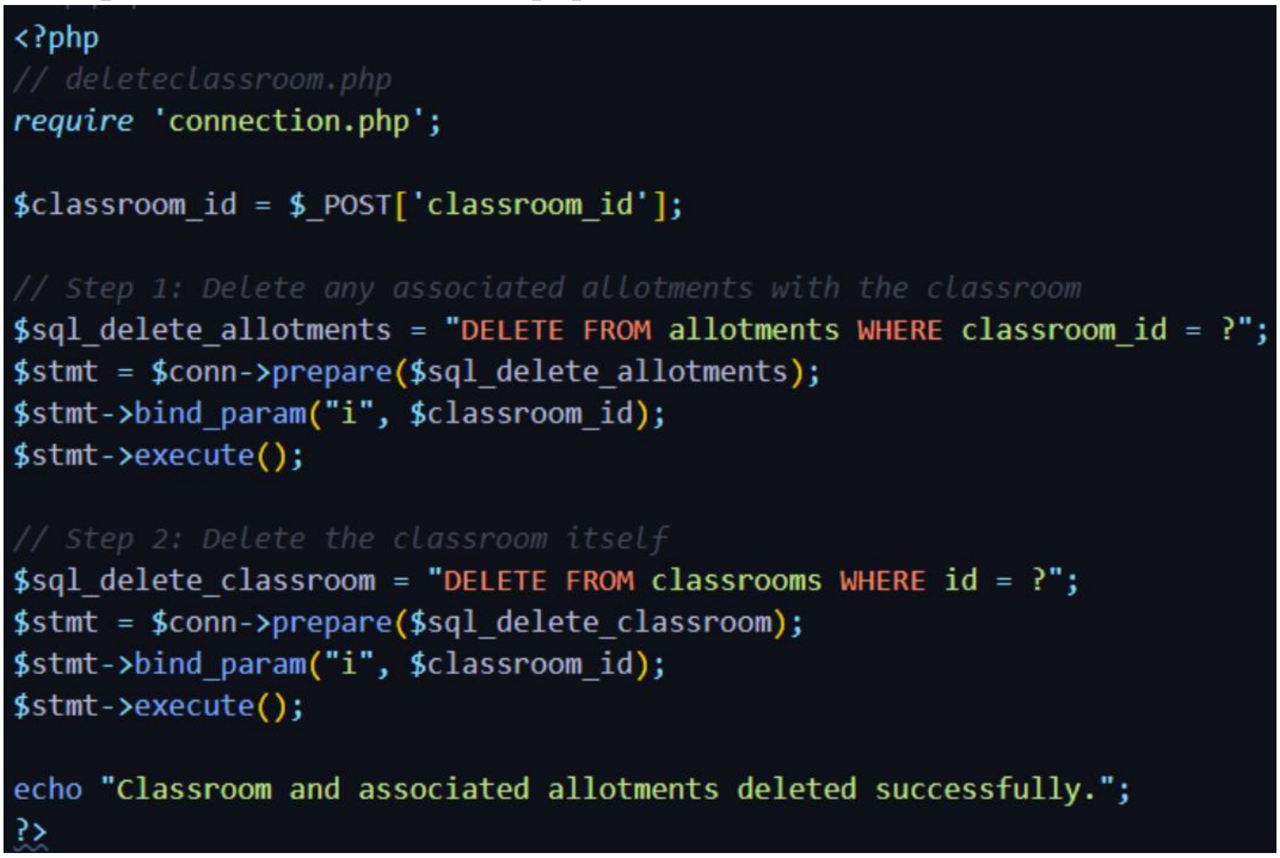


```php
<?php
// addclassrooms.php
require 'connection.php';

if ($_SERVER['REQUEST_METHOD'] == 'POST') {
    $classroom_name = $_POST['classroom_name'];
    $capacity = $_POST['capacity'];

    // Insert classroom into the database
    $sql = "INSERT INTO classrooms (name, capacity) VALUES (?, ?)";
    $stmt = $conn->prepare($sql);
    $stmt->bind_param("si", $classroom_name, $capacity);
    $stmt->execute();
    echo "Classroom added successfully.";
}
?>
```

- Functional cohesion is achieved when a file or function performs a single, well-defined task. Many of your files in the Course Scheduler project exhibit this type of cohesion, where each file has one clear responsibility.
- In this file, addclassrooms.php solely focuses on adding classrooms, which makes it highly cohesive. This makes the file easy to understand, maintain, and test since it has a single, dedicated responsibility.

2. SEQUENTIAL COHESION

Example Files: deleteclassroom.php



```php
<?php
// deleteclassroom.php
require 'connection.php';

$classroom_id = $_POST['classroom_id'];

// Step 1: Delete any associated allotments with the classroom
$sql_delete_allotments = "DELETE FROM allotments WHERE classroom_id = ?";
$stmt = $conn->prepare($sql_delete_allotments);
$stmt->bind_param("i", $classroom_id);
$stmt->execute();

// Step 2: Delete the classroom itself
$sql_delete_classroom = "DELETE FROM classrooms WHERE id = ?";
$stmt = $conn->prepare($sql_delete_classroom);
$stmt->bind_param("i", $classroom_id);
$stmt->execute();

echo "Classroom and associated allotments deleted successfully.";
?>
```

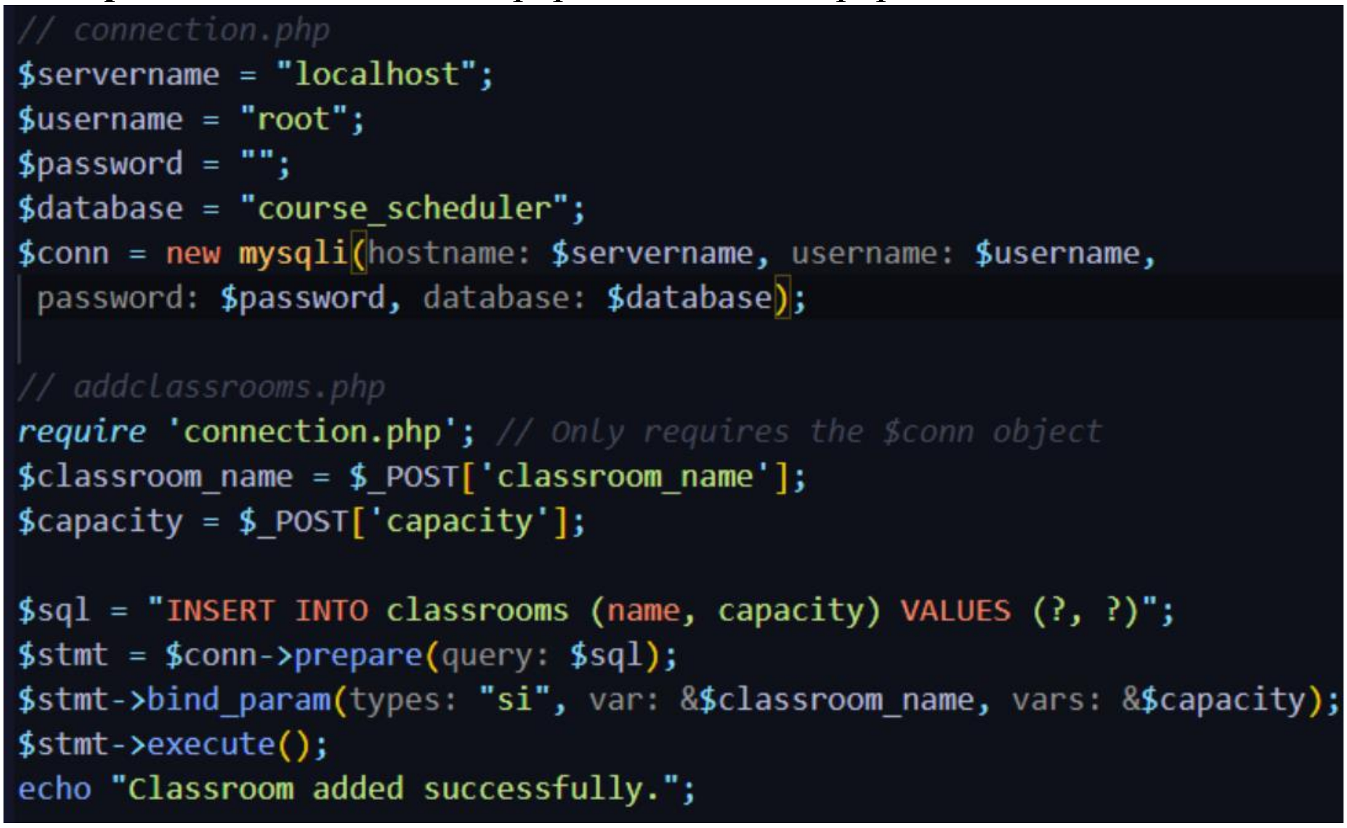
- Sequential cohesion occurs when one task follows logically after another, where the output of one step is used as input for the next. This type of cohesion is typically present in files that perform a series of dependent tasks
- Here, deleteclassroom.php sequentially performs operations—first deleting related records, then deleting the classroom. Each step logically depends on the previous one, enhancing cohesion within the file.

Coupling:

In the Course Scheduler, we designed modules to have loose coupling by minimizing dependencies and interactions between them. This ensures that changes in one module don't require extensive changes in others, which enhances the maintainability and scalability of the application.

1. Data Coupling

Example Files: addclassrooms.php and connection.php

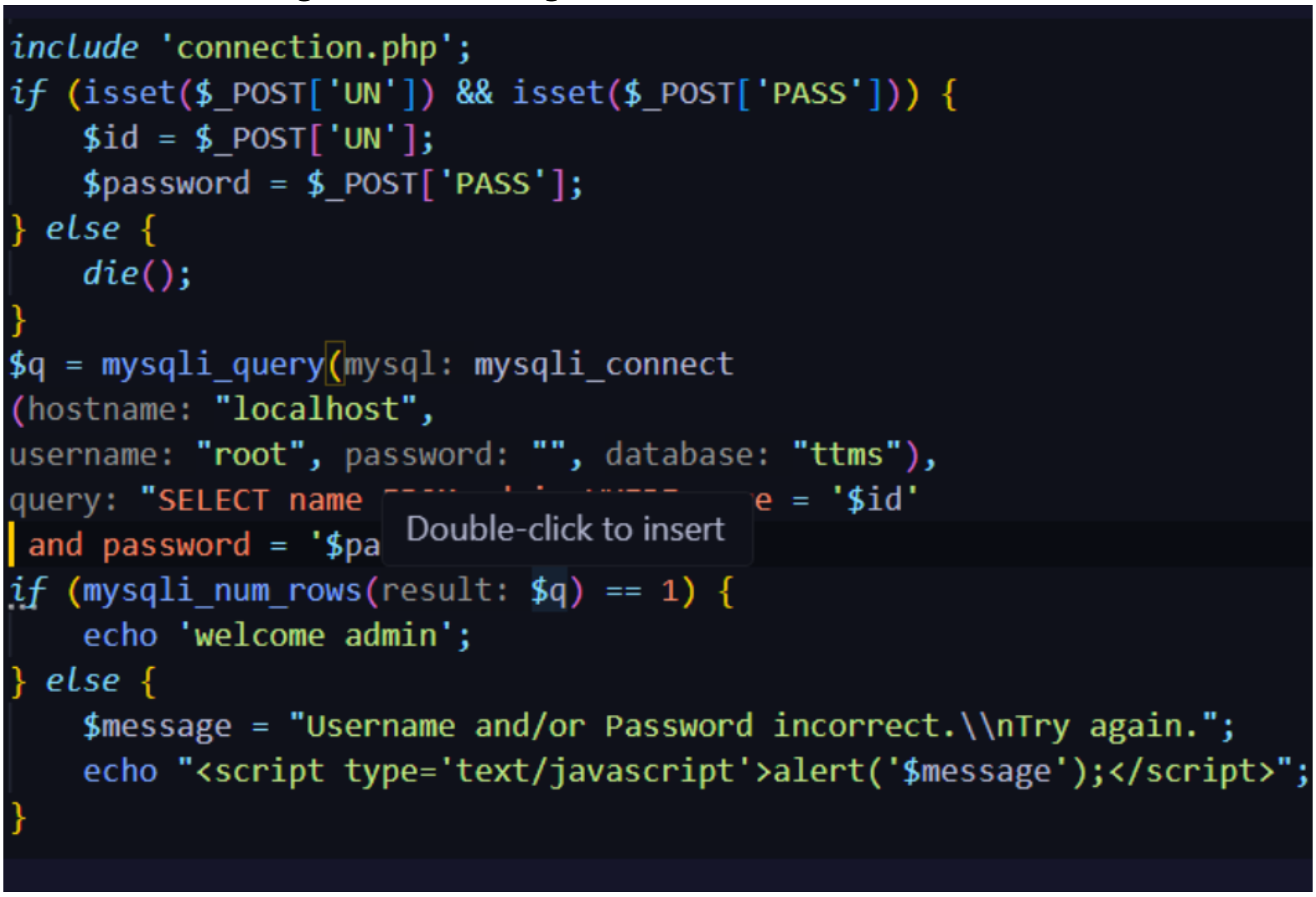


```php
// connection.php
$servername = "localhost";
$username = "root";
$password = "";
$database = "course_scheduler";
$conn = new mysqli(hostname: $servername, username: $username,
 password: $password, database: $database);

// addclassrooms.php
require 'connection.php'; // Only requires the $conn object
$classroom_name = $_POST['classroom_name'];
$capacity = $_POST['capacity'];

$sql = "INSERT INTO classrooms (name, capacity) VALUES (?, ?)";
$stmt = $conn->prepare(query: $sql);
$stmt->bind_param(types: "si", var: &$classroom_name, vars: &$capacity);
$stmt->execute();
echo "Classroom added successfully.";
```

- Data coupling occurs when two modules share data, typically by passing simple data values (like numbers or strings) between functions or files without sharing internal implementation details. This is a loose form of coupling, as it only requires the necessary data to be shared, without making modules dependent on each other's implementation.
- In this case, addclassrooms.php only depends on the $conn object from connection.php, which represents data coupling.

2. Stamp Coupling

Example Files: assignSubstituteForm.php might receive a composite data structure containing substitute assignment details.



```
include 'connection.php';
if (isset($_POST['UN']) && isset($_POST['PASS'])) {
    $id = $_POST['UN'];
    $password = $_POST['PASS'];
} else {
    die();
}
$q = mysqli_query(mysql: mysqli_connect
(hostname: "localhost",
username: "root", password: "", database: "ttms"),
query: "SELECT name [illegible]e = '$id'
 and password = '$pa[illegible]
if (mysqli_num_rows(result: $q) == 1) {
    echo 'welcome admin';
} else {
    $message = "Username and/or Password incorrect.\\nTry again.";
    echo "<script type='text/javascript'>alert('$message');</script>";
}
```

- Stamp coupling (also known as structure coupling) occurs when modules share a composite data structure, like an object or a record, instead of individual data elements. This type of coupling is useful when multiple pieces of related data need to be passed together, but it also increases dependency because the receiving module must understand the structure of the data.

- Here, the function assignSubstitute relies on the structure substituteDetails, which increases dependency on its layout, representing stamp coupling.