**Name:** Aljon M. Viñas
**Year and Section:** BSIS 2A

## Scenario 1 — Using $_POST instead of $_GET

```
<?php
$conn = mysqli_connect("localhost", "root", "", "class_db");
# Switch POST to GET
# We are using GET because we want to send the data in the URL
# GET is used to get data from the URL
$id = $_GET['id'];
$sql = "SELECT * FROM students WHERE id = $id";
$res = mysqli_query($conn, $sql);
$r = mysqli_fetch_assoc($res);
echo $r['first_name'];
?>
```

## Scenario 2 — Missing quotes in SQL when using POST

```
<?php
$conn = mysqli_connect("localhost","root","","class_db");
$fname = $_POST['fname'];
# Don't put $fname inside quotes in the SQL query
# If we don't use ' ' around $fname, it will cause an error
# We use ' ' because $fname is a string, not a number
$sql = "SELECT * FROM students WHERE first_name = '$fname' ";
$res = mysqli_query($conn, $sql);
?>
```

## Scenario 3 — SQL injection vulnerability

```
<?php
$conn = mysqli_connect("localhost","root","","class_db");
# Use prepared statements to protect against SQL injection
# Directly inserting user input into SQL can be unsafe
$stmt = $conn->prepare("SELECT * FROM students WHERE age = ?");
$stmt->bind_param("i", $age);
$stmt->execute();
?>
```

## Scenario 4 — Forgetting to validate empty POST field

```
<?php
$conn = mysqli_connect("localhost","root","","class_db");
# We check that both inputs are filled before adding data
# This stops empty values from being saved in the database
if (!empty($_POST['fname']) && !empty($_POST['lname'])) {
   $first = $_POST['fname'];
   $last = $_POST['lname'];
   # Add the data only when both names are provided
   $sql = "INSERT INTO students (first_name, last_name) VALUES ('$first', '$last')";
   mysqli_query($conn, $sql);
   echo "Inserted!";
} else {
   # Runs if one or both inputs are missing
   echo "Please fill out both first and last name.";
}
?>
```

## Scenario 5 — Wrong key name in POST

```
<?php
$conn = mysqli_connect("localhost","root","","class_db");
# The POST key was written wrong earlier, so we changed it to 'email'
$email = $_POST['email'];
$sql = "SELECT * FROM students WHERE email='$email'";
$res = mysqli_query($conn, $sql);
?>
```

## Scenario 6 — Unsafe direct use of GET in DELETE

```
<?php
$conn = mysqli_connect("localhost","root","","class_db");
# Change the value to a number so harmful input cannot be used
$id = intval($_GET['id']);
$sql = "DELETE FROM students WHERE id = $id";
mysqli_query($conn, $sql);
?>
```

## Scenario 7 — Query fails but script continues

```
<?php
$conn = mysqli_connect("localhost","root","","class_db");
$id = $_POST['id'];
$email = $_POST['email'];
# We put quotes around the email because it is text, not a number
# We also check for errors so it doesn't say “Updated!” when the query fails
$sql = "UPDATE students SET email='$email' WHERE id=$id";
if (!$res = mysqli_query($conn, $sql)) {
   echo "Error updating!";
}
?>
```

## Scenario 8 — Missing mysqli_fetch_assoc loop

```
<?php
$conn = mysqli_connect("localhost","root","","class_db");
$res = mysqli_query($conn,"SELECT * FROM students");
# We use a while loop to get every record
# Instead of only getting one row
while ($row = mysqli_fetch_assoc($res)) {
   echo $row['email'] . "<br>";
}
?>
```

## Scenario 9 — Using GET but link sends POST

```
<?php
# Links send data with GET, not POST
# POST is mainly used when sending form data
$id = $_GET['id'];
?>
<a href="view.php?id=3">View Student</a>
```

## Scenario 10 — Wrong variable used in SQL

```
<?php
$age = $_POST['age'];
# The variable was misspelled as 'aeg' before
# Fixed it to the correct name 'age'
$sql = "SELECT * FROM students WHERE age = $age";
```

```
$res = mysqli_query($conn, $sql);

?>
```

## Scenario 11 — Mismatched method (expects POST but form sends GET)

```
<?php

# Get the email value from the URL

$email = $_GET['email'];

?>
```

## Scenario 12 — Numeric GET used inside quotes

```
<?php

$id = $_GET['id'];

# ID is a number, so we don't put it inside quotes

$sql = "SELECT * FROM students WHERE id = $id";

?>
```

## Scenario 13 — Missing WHERE clause in UPDATE

```
<?php

$newEmail = $_POST['email'];

# We add a WHERE clause so only one student is updated

# Without it, all rows in the table would change

$sql = "UPDATE students SET email='$newEmail' WHERE student_id=$id";

mysqli_query($conn, $sql);

?>
```

## Scenario 14 — Using POST array incorrectly

```
<?php

$data = $_POST;

# Make sure the array keys are correct

# Put string values inside quotes in the SQL query

$sql = "INSERT INTO students (first_name, last_name, email)

   VALUES ('{$data['first_name']}', '{$data['last_name']}', '{$data['email']}')";

?>
```

## Scenario 15 — GET parameter used inside SQL without sanitization

```
<?php

# Get the page number from the URL

$page = $_GET['page'];

# Change it to a number to block text or symbols

$page = intval($page);

# Make sure the page number is not negative
```

```
if ($page < 0) {
    $page = 0;
}
# Pagination setup
$limit = 5;          # Number of records per page
$offset = $page * $limit;  # Start point for the query
# SQL query to get the current page of students
$sql = "SELECT * FROM students LIMIT $offset, $limit";
?>
```