REPUBLIC OF THE PHILIPPINES
BICOL UNIVERSITY
POLANGUI
Polangui, Albay

Email: bupc-dean@bicol-u.edu.ph

ISO 9001: 2015
SOCOTEC SCP000722Q

BAGONG PILIPINAS

NAME: JUAREZ, ANNALIZA N.
COURSE, YEAR & SECTION: BSIS 2A

SUBJECT: WEB SYSTEM
PROFESSOR: Reymar Llagas

SCENARIO 1 — Using $_POST instead of $_GET

PROBLEM: Using $_POST['id'] when the value comes from the URL (?id=3).

SOLUTION: Use $_GET['id'] to get the value from the URL.

EXPLANATION: Kasi yung value ng `id` galing sa URL (`?id=3`), kailangan gamitin ang `$_GET` para makuha ito nang tama sa PHP at maiwasan yung error na "Undefined index."

Scenario 2 — Missing quotes in SQL when using POST

PROBLEM: The SQL query is missing quotes around the string, so MySQL interprets the input (Ana) as a column name.

SOLUTION: Put string values in quotes: WHERE first_name = '$fname'.

EXPLANATION: Dahil string ang galing sa form (fname), kailangan nakalagay sa quotes para maintindihan ng SQL na value ito at hindi column, para maiwasan ang "Unknown column" error.

Scenario 3 — : SQL injection vulnerability

PROBLEM: The SQL query directly uses $_GET['age'] without validation, allowing attackers to inject malicious input like 1 OR 1=1, which returns all records.

SOLUTION: Use a prepared statement with ? placeholder and bind the parameter.

EXPLANATION: Kapag diretsong nilalagay sa query ang value galing sa URL, puwedeng ma- SQL inject pero kapag prepared statement, nalilinis ang input kaya hindi nagiging code at mas safe ang database.

Scenario 4 — Forgetting to validate empty post field

PROBLEM: The form inputs are not checked, so submitting empty fields can insert blank rows or cause SQL errors.

SOLUTION: Validate inputs and then insert: INSERT INTO students (first_name, last_name) VALUES ('$first', '$last').

EXPLANATION: Dahil walang laman ang form, puwedeng ma- insert ang blank na rows sa database; kapag naglagay ng validation, siguradong may laman ang first at last name bago ito i-save.

Scenario 5 — Wrong key name in POST

PROBLEM: The POST key is misspelled (emial instead of email), causing an "Undefined index" error.

SOLUTION: Use the correct POST key: $_POST['email']

EXPLANATION: Kasi maling spelling ng key sa POST, hindi niya mahahanap yung value; kapag ginamit ang tamang email, makukuha ng PHP yung data mula sa form.

Scenario 6 — Unsafe direct use of GET in DELETE

PROBLEM: The code directly inserts $_GET['id'] into the DELETE query, allowing dangerous inputs like ?id=0 OR 1=1 that can delete all records.

SOLUTION: Sanitize ID with intval($_GET['id']) before DELETE.

EXPLANATION: Kung diretsong ginamit ang id galing sa URL, puwedeng magpasok ang user ng malisyosong query; gamit ang prepared statement, siguradong number lang ang matatanggap at safe ang data.

Scenario 7 — Query fails but script continues

PROBLEM: Missing quotes around the email value, so the UPDATE query fails, but the script still prints "Updated!".

SOLUTION: Include WHERE in UPDATE: UPDATE students SET email='$email' WHERE student_id=$id.

EXPLANATION: Walang quotes sa email, nag-error ang query pero nag- print pa rin ng "Updated!"; kapag may quotes at nag- check ng result, makikita mo kung nag- success talaga ang update.

Scenario 8 — Missing mysqli_fetch_assocloop

PROBLEM: Only the first row is fetched, so only one email is displayed.

SOLUTION: Use a while loop to fetch all rows: while ($row = mysqli_fetch_assoc($res))

EXPLANATION: Dahil isang beses lang kinukuha ang row, isang email lang ang lumalabas; kapag ginamit ang `while` loop, lahat ng emails sa database ay ma- display.

Scenario 9 — Using GET but link sends POST
PROBLEM: PHP is using $_POST['id'], but the link sends id via URL (GET), causing “Undefined index” error.
SOLUTION: Change to use $_GET
EXPLANATION: Kasi ang link nagse-send ng `id` sa URL, dapat `$_GET` ang gamitin para makuha ang value at maiwasan ang error.

Scenario 10 — Wrong variable used in SQL
PROBLEM: The SQL query uses $aeg instead of the correct $age, causing “Undefined variable” error.
SOLUTION: Correct variable name: $age = $_POST['age'].
EXPLANATION: Maling variable ang ginamit sa query, hindi niya makita ang value; kapag tama ang variable ($age), gumagana nang maayos ang SQL.

Scenario 11 — Mismatched method (expects POST but form sends GET)
PROBLEM: The form sends data via GET, but PHP is trying to read it using $_POST['email'], causing “Undefined index” error.
SOLUTION: Change PHP to use $_GET:
EXPLANATION: Dahil ibang method ang gamit ng form at PHP, hindi niya makita ang value; kung i-match mo ang method, makukuha na ng PHP ang email.

Scenario 12 — Numeric GET used inside quotes
PROBLEM: The ID is numeric, but it’s wrapped in quotes in SQL, which can cause inefficiency or wrong index usage.
SOLUTION: Convert numeric GET to integer: $id = intval($_GET['id'])
EXPLANATION: Kasi number ang ID, di na kailangan ng quotes; kapag ginawang integer, mas mabilis at tama ang query.

Scenario 13 — Missing WHERE clause in UPDATE
PROBLEM: The UPDATE query has no WHERE clause, so it updates all rows in the table.
SOLUTION: Always include WHERE in UPDATE: WHERE student_id=$id.
EXPLANATION: Dahil WHERE, lahat ng rows ang na- update; kapag may WHERE, isang record lang ang mababago.

Scenario 14 — Using POST array incorrectly
PROBLEM: The POST array values are not properly quoted, causing “Undefined index” or SQL errors.
SOLUTION: Reference POST array correctly: {$data['first_name']}
EXPLANATION: Kasi mali ang paraan ng pagkuha ng values sa array at walang quotes, nagkaka-error; kapag tama ang reference at may quotes, maayos na ma- insert sa database.

Scenario 15 — GET parameter used inside SQL without sanitization
PROBLEM: The page number from $_GET['page'] is used directly in the SQL LIMIT, making it vulnerable to invalid or malicious input.
SOLUTION: Validate pagination input: $page = intval($_GET['page']); if($page<0) $page=0
EXPLANATION: Kung hindi nililinis ang page number, puwedeng maglagay ang user ng maling value at mag-cause ng error, pero pag ginamit ang intval(), siguradong number lang ang papasok.
