Name: Cornal, Arlezar G. **Subject:** WEB SYSTEM

Course & Block: BSIS – **2** **Professor:** Reymark, Llagas

Scenario	Problem	Correct Answer
1	Used $_POST even though the ID comes from the URL.	Use $_GET and check if it exists.
2	Missing quotes around a string in SQL.	Put the first name inside quotes.
3	SQL injection risk using raw $_GET['age'].	Use a prepared statement to bind the age.
4	Blank fields get inserted because there's no validation.	Check if first and last name are not empty before inserting.
5	Typo in POST key (emial).	Use the correct key email.
6	Unsafe DELETE uses raw GET input.	Convert ID to an integer or use a prepared statement.
7	No quotes around email + no error checking.	Add quotes and check if the query succeeded.
8	Only one row fetched because no loop.	Use a loop to fetch all rows.
9	Using POST but link sends GET.	Change to $_GET.
10	Wrong variable name ($aeg).	Correct it to $age.
11	Form sends GET but PHP expects POST.	Make either the form or PHP use the same method.
12	Numeric ID placed inside quotes.	Remove quotes or cast the ID to an integer.
13	UPDATE statement has no WHERE clause.	Add a WHERE condition to update a specific student.
14	Incorrect usage of POST array; missing quotes.	Use proper array indexing and treat values as strings.
15	Unvalidated page number allows invalid or huge values.	Convert page to integer and restrict negative/invalid numbers.

EXPLANATION:
Scenario 1 May bug yung code which is naga expect siya na post, but ang value na tig send via get the url, so we need to use the $_get in order to read perfectly yung parameter

Scenario 2 Sa sql, dapat naka quote talaga ang string. Dahil pag hindi mo i qoute. The sql gonna think na column name sya kaya ang lumalabas yung “unknown column” issue.

Scenario 3 Pag directo mo na sinasaksak yung GET values sa SQL, super risky yan for SQL injection. Prepared statements ang nag-e-ensure na safe and properly sanitized yung input bago siya ma-run. When you directly plug yung get values sa sql, super risk yan for sql injection. So the prepared statement ang nage-ensure na sade and properly sanitized yung input bago sya ma-run

Scenario 4 Kun may blank data na sinasubmit, puwedeng maka-mess up sa database. Kaya importante na mag-validate ka muna bago mag-INSERT.

Scenario 5 Pag wrong spelling yung POST key, hundi makakuha si PHP ang value, kaya may undefined index. Ayusin lang yung key para ma-capture nang maayos yung email.

Scenario 6 Pag raw GET value ang ginamit sa DELETE query, pwede niya i-delete ang unlimited records. Kung i-cast mo siya as integer, sure na isang ID lang ang pwedeng mada-delete.

Scenario 7 Kahit nag-fail na yung SQL, nagsasabi pa rin yung script na “Updated!” kasi hindi siya nag-che-check ng errors. Kailangan talaga mag-add ng proper error handling para ma-report ng sakto.

Scenario 8 Ang mysqli_fetch_assoc once lang mag-fetch kung hindi nasa loop. Kaya kailangan mo talaga i-while loop para lumabas ang lahat na records.

Scenario 9 Yung link naga-trigger ng GET request, pero yung script magre-read ng POST, kaya may error. Gamiton mo $_GET para match sa behavior.

Scenario 10 Pag mali yung variable name, nag-wa-warning si PHP tapos nababali pa yung SQL query. I-fix lang yung name para ma-use ang right value.

Scenario 11 Yung form nagse-send ng GET, pero yung PHP code nage-expect ng POST. Dapat pareho sila—both GET or both POST—para makuha talaga ang data.

Scenario 12 Ang ID dapat number, so di siya dapat naka-quote. Mas klaro at mas safe kung i-cast as integer.

Scenario 13 Pag walang WHERE clause, ma-u-update yung entire table. Dapat may WHERE para specific lang yung record na ma-update.

Scenario 14 Yung array keys dapat properly enclosed, tapos yung string values dapat naka-quote. Pag tama pareho, ok na ang SQL query.

Scenario 15 Kun sobra ka-laki yung page number na sinasabi, puwede mag-generate ng huge offset para humina or maka-crash sa database. Kaya kailangan may limit and validation.