

# **Lab Assessment – 4**

## **Name:** Hitharth Rajani

## **Reg. No. :** 23BCB0097

## **Course Name & Code :** Database Systems Lab (BCSE302P)

## **Slot :** L27+L28

## **Faculty Name:** Prof. Saurabh Agarwal

**Question 1: In your project, design and implement a stored procedure that automates a key operation (e.g., updating student grades, generating monthly sales reports, or calculating total bill amount). Explain how you used functions and cursors within the procedure to retrieve and process multiple records efficiently.**

INSERT INTO Bookings (user\_id, vehicle\_id, slot\_id, start\_time, end\_time, total\_cost, booking\_status) VALUES

(1, 1, 1, '2025-09-10 08:00:00', '2025-09-10 11:00:00', 15.00, 'Completed'),

(1, 1, 2, '2025-09-15 14:00:00', '2025-09-15 18:00:00', 20.00, 'Completed'),

(2, 2, 3, '2025-09-20 10:00:00', '2025-09-20 12:00:00', 7.00, 'Completed');

DELIMITER $$

CREATE PROCEDURE GenerateUserMonthlyReport(

IN p\_user\_id INT,

IN p\_report\_month INT,

IN p\_report\_year INT

)

BEGIN

DECLARE v\_booking\_id INT;

DECLARE v\_booking\_cost DECIMAL(10, 2);

DECLARE v\_monthly\_total\_cost DECIMAL(10, 2) DEFAULT 0.00;

DECLARE done INT DEFAULT FALSE; -- Loop control variable

DECLARE booking\_cursor CURSOR FOR

SELECT booking\_id, total\_cost

FROM Bookings

WHERE user\_id = p\_user\_id

AND MONTH(end\_time) = p\_report\_month

AND YEAR(end\_time) = p\_report\_year

AND booking\_status = 'Completed';

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN booking\_cursor;

read\_loop: LOOP

FETCH booking\_cursor INTO v\_booking\_id, v\_booking\_cost;

IF done THEN

LEAVE read\_loop;

END IF;

SET v\_monthly\_total\_cost = v\_monthly\_total\_cost + v\_booking\_cost;

END LOOP;

CLOSE booking\_cursor;

SELECT

p\_user\_id AS user\_id,

p\_report\_month AS report\_month,

p\_report\_year AS report\_year,

v\_monthly\_total\_cost AS total\_monthly\_cost,

(SELECT COUNT(\*) FROM Bookings WHERE user\_id = p\_user\_id AND MONTH(end\_time) = p\_report\_month AND YEAR(end\_time) = p\_report\_year AND booking\_status = 'Completed') AS total\_bookings;

END$$

DELIMITER ;

**1.Functions:** The procedure achieves efficiency by using functions for broad, fast filtering and a cursor for detailed, focused processing. Built-in functions like MONTH() and YEAR() are used in the WHERE clause to immediately narrow down the entire Bookings table to only the records from the specified month and year. This is a highly optimized, set-based operation that ensures the next step has a much smaller dataset to work with. Similarly, the COUNT() function is used to get the total number of bookings in a single, efficient query, avoiding the need for a manual count.

**2. Cursors:** Once the data is filtered, the cursor takes over to handle the records one by one. Instead of iterating through the entire table, the cursor only needs to loop through the small, pre-filtered result set. Inside the loop, the FETCH command retrieves each booking's cost, allowing for procedural logic like adding it to a running total. This synergistic approach—using functions for high-performance filtering and a cursor for row-level processing on the small result—ensures the entire operation is both fast and scalable.

**Question 2: Create a trigger in your project database that ensures data integrity (e.g., preventing deletion of a student with pending fees or updating stock quantity automatically after a new sale).Discuss how the trigger interacts with existing procedures and functions in your project to maintain consistent system behaviour.**

DELIMITER $$

CREATE TRIGGER PreventUserDeletionWithActiveBookings

BEFORE DELETE ON Users

FOR EACH ROW

BEGIN

DECLARE active\_booking\_count INT;

SELECT COUNT(\*)

INTO active\_booking\_count

FROM Bookings

WHERE user\_id = OLD.user\_id

AND booking\_status IN ('Reserved', 'Active');

IF active\_booking\_count > 0 THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Cannot delete user: This user has active or reserved bookings.';

END IF;

END$$

* The **PreventUserDeletionWithActiveBookings**trigger serves as a foundational layer of data integrity, interacting indirectly but critically with the **CreateBooking** procedure. Its primary role is to intercept any attempt to delete a user and check for any 'Active' or 'Reserved' bookings associated with them. By preventing the deletion if such bookings exist, the trigger guarantees that the valid state created by the **CreateBooking** procedure cannot be logically corrupted. It ensures that no booking record can become orphaned, thereby maintaining a consistent and reliable relationship between users and their ongoing parking sessions at the most fundamental level.
* This enforcement of integrity has a direct positive impact on reporting and data retrieval functions. For instance, the **GenerateUserMonthlyReport** procedure relies on the assumption that every booking is linked to a valid user. The trigger upholds this assumption, preventing potential errors or inaccurate calculations that would arise from orphaned records. Similarly, views like V\_BookingDetails, which join the Users and Bookings tables, are protected from displaying inconsistent data, such as NULL values for a user who was improperly deleted. In essence, the trigger provides a stable and predictable data environment, allowing higher-level procedures and views to operate with the confidence that the underlying data is always valid.

**Text in bold above are procedures.**