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.

```sql
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.