DBMS: LAB 6
TRIGGERS, FUNCTIONS AND PROCEDURE Commands
University Fest Management System

Name: M NIRANJAN
SRN:PES2UG23CS308
SECTION: E

OBJECTIVE: To learn and understand Triggers, Procedures, and Functions while executing queries in MySQL.

TRIGGERS

- Automatically execute SQL statements in response to INSERT, UPDATE, or DELETE events on a table.

PROCEDURES

- A set of SQL statements grouped together, executed using CALL, to perform tasks or operations.

FUNCTIONS

- Similar to procedures but must return a single value and can be used directly inside SQL queries

---

INSTRUCTIONS:

As a part of LAB 6, there are 3 tasks to be completed with respect to the case study shared earlier with the ER diagram and Relational Schema of University Fest Management:

- TASK 1: Create and implement Triggers to automatically perform actions on table events (INSERT, UPDATE, DELETE).
- TASK 2: Create and implement Functions to perform calculations or return values that can be used in SQL queries.
- TASK 3: Create and implement Procedures to group SQL statements and execute them using CALL for performing operations or tasks.

● As a part of the submission process, the following are to be submitted:

- A PDF document, containing all the Screenshots for all three tasks as suggested
  ▪ Name of the file: <your SRN>_University_Fest_DB_Lab6.pdf
- The ".sql" file for the same, shall contain all the commands that have been executed in the lab
  ▪ Name of the file: <your SRN>_University_Fest_DB_Lab6.sql

Example:
Refer to the sample submissions given below. This will give you an idea about the details that must be included in your submissions.

NOTE: Screenshots can be taken either from **MySQL Workbench** or **Command Line**.

For every task (Trigger, Function, Procedure), **3 screenshots are required**:

1. **Definition** of the Trigger/Function/Procedure (CREATE statement).
2. **Execution** of the Trigger/Function/Procedure (INSERT/UPDATE/CALL/SELECT as applicable).
3. **Result/Output** after execution (showing the effect or returned value).

Sample submission:

BEFORE:

COMMAND:

AFTER:

TASK 1 [TRIGGERS]:

1. Create a trigger that automatically decreases the total_quantity of an item in the stall_items table whenever a new row is inserted into the purchased table. (hint: Use AFTER INSERT on purchased, and update the corresponding record in stall_items.)

Before: --describe the table/s

```
mysql> SELECT * FROM stall_items WHERE stall_id='S1' AND item_name='Veggie W
rap';
+----------+------------+---------------+---------------+
| stall_id | item_name  | price_per_unit | total_quantity |
+----------+------------+---------------+---------------+
| S1       | Veggie Wrap |        399.00 |            50 |
+----------+------------+---------------+---------------+
1 row in set (0.02 sec)
```

Command – sql command

```
mysql> DELIMITER //
mysql> CREATE TRIGGER decrease_quantity_after_purchase
    -> AFTER INSERT ON purchased
    -> FOR EACH ROW
    -> BEGIN
    ->     UPDATE stall_items
    ->     SET total_quantity = total_quantity - NEW.quantity
    ->     WHERE stall_id = NEW.stall_id
    ->       AND item_name = NEW.item_name;
    -> END;
    -> //
ERROR 1359 (HY000): Trigger already exists
mysql> DELIMITER ;
mysql>
mysql> -- Test: insert a new purchase
mysql> INSERT INTO purchased VALUES('P1002','S1','Veggie Wrap',NOW(),2);
Query OK, 1 row affected (0.05 sec)
```

After: --describe the table/s

```
mysql> SELECT * FROM stall_items WHERE stall_id='S1' AND item_name='Veggie W
rap';
+----------+-------------+----------------+----------------+
| stall_id | item_name   | price_per_unit | total_quantity |
+----------+-------------+----------------+----------------+
| S1       | Veggie Wrap |         399.00 |             48 |
+----------+-------------+----------------+----------------+
1 row in set (0.00 sec)
```

2. Create a trigger that prevents a participant from purchasing more than 5 units of any single item in a single transaction (i.e., quantity > 5) in the purchased table.
(hint: Use BEFORE INSERT and raise an error if NEW.quantity > 5.)

Before: --describe the table/s

```
mysql> DESC purchased;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| SRN       | varchar(10) | NO   | PRI | NULL    |       |
| stall_id  | varchar(5)  | NO   | PRI | NULL    |       |
| item_name | varchar(25) | NO   | PRI | NULL    |       |
| timestamp | timestamp   | NO   | PRI | NULL    |       |
| quantity  | int         | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
5 rows in set (0.02 sec)
```

Command – sql command

```
mysql> DELIMITER //
mysql> CREATE TRIGGER prevent_large_purchase
    -> BEFORE INSERT ON purchased
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.quantity > 5 THEN
    ->         SIGNAL SQLSTATE '45000'
    ->         SET MESSAGE_TEXT = 'Cannot purchase more than 5 units in a si
ngle transaction.';
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;
mysql>
mysql> -- Test: try invalid insert
mysql> INSERT INTO purchased VALUES('P1003','S2','Fish Tacos',NOW(),6);
```

After: --describe The table

```
ERROR 1644 (45000): Cannot purchase more than 5 units in a single transactio
n.
```

TASK 2 [PROCEDURES]:

1. Write a **stored procedure** GetStallSales that takes a stall_id as input and prints the **total revenue** generated from that stall based on the purchased table and item prices from stall_items.
   (hint: Join purchased with stall_items and calculate SUM(price_per_unit * quantity))

Before: --describe the table/s

```
mysql> SELECT * FROM purchased WHERE stall_id='S1';
+-------+----------+--------------------------+---------------------+------
----+
| SRN   | stall_id | item_name                | timestamp           | quant
ity |
+-------+----------+--------------------------+---------------------+------
----+
| P1001 | S1       | Mushroom Risotto         | 2023-04-15 13:00:05 |
  3 |
| P1001 | S1       | Veggie Wrap              | 2023-04-15 12:00:00 |
  2 |
| P1002 | S1       | Caprese Salad            | 2023-04-16 13:33:00 |
  3 |
| P1002 | S1       | Classic Caesar Salad     | 2023-04-16 13:33:00 |
  3 |
| P1002 | S1       | Veggie Wrap              | 2025-09-17 17:44:42 |
  2 |
| P1006 | S1       | Margherita Pizza         | 2023-04-16 14:50:00 |
  1 |
| P1017 | S1       | Classic Caesar Salad     | 2023-04-16 10:05:00 |
  3 |
| P1017 | S1       | Spinach and Feta Omelette | 2023-04-16 10:05:00 |
  2 |
+-------+----------+--------------------------+---------------------+------
----+
8 rows in set (0.00 sec)

mysql> SELECT * FROM stall_items WHERE stall_id='S1';
+----------+--------------------------+----------------+----------------+
| stall_id | item_name                | price_per_unit | total_quantity |
+----------+--------------------------+----------------+----------------+
| S1       | Caprese Salad            |         249.00 |             35 |
| S1       | Classic Caesar Salad     |         349.50 |             45 |
| S1       | Margherita Pizza         |         350.00 |             25 |
| S1       | Mushroom Risotto         |         359.00 |             25 |
| S1       | Spinach and Feta Omelette |         259.00 |             40 |
| S1       | Vegetable Pad Thai       |         329.00 |             25 |
| S1       | Vegetable Stir-Fry       |         299.00 |             30 |
| S1       | Veggie Wrap              |         399.00 |             48 |
+----------+--------------------------+----------------+----------------+
```

Command – sql command and After: --describe the table/s

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetStallSales(IN input_stall_id VARCHAR(5))
    -> BEGIN
    ->     SELECT s.stall_id, SUM(si.price_per_unit * p.quantity) AS total_r
evenue
    ->     FROM purchased p
    ->     JOIN stall_items si
    ->       ON p.stall_id = si.stall_id AND p.item_name = si.item_name
    ->     JOIN stall s
    ->       ON s.stall_id = p.stall_id
    ->     WHERE s.stall_id = input_stall_id
    ->     GROUP BY s.stall_id;
    -> END;
    -> //
Query OK, 0 rows affected (0.03 sec)

mysql> DELIMITER ;
mysql>
mysql> -- Call
mysql> CALL GetStallSales('S1');
+----------+---------------+
| stall_id | total_revenue |
+----------+---------------+
| S1       |       6385.00 |
+----------+---------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

2. Create a procedure RegisterParticipant that registers a participant (SRN) for an event (event_id) by inserting into the registration table. The procedure should take the event_id, SRN, and registration_id as parameters.
   (hint: Use input parameters and basic INSERT.)

Before: --describe the table/s

```
mysql> SELECT * FROM registration WHERE event_id='E1';
+----------+-------+-----------------+
| event_id | SRN   | registration_id |
+----------+-------+-----------------+
| E1       | P1017 | R2              |
| E1       | P1022 | R3              |
| E1       | P1025 | R4              |
+----------+-------+-----------------+
3 rows in set (0.01 sec)
```

Command – sql command

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE RegisterParticipant(
    ->     IN input_event_id VARCHAR(5),
    ->     IN input_SRN VARCHAR(10),
    ->     IN input_regid VARCHAR(5)
    -> )
    -> BEGIN
    ->     INSERT INTO registration(event_id, SRN, registration_id)
    ->     VALUES (input_event_id, input_SRN, input_regid);
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
mysql> -- Call
mysql> CALL RegisterParticipant('E1','P1011','R200');
Query OK, 1 row affected (0.01 sec)
```

After: --describe the table/s

```
mysql> SELECT * FROM registration WHERE event_id='E1';
+----------+-------+-----------------+
| event_id | SRN   | registration_id |
+----------+-------+-----------------+
| E1       | P1011 | R200            |
| E1       | P1017 | R2              |
| E1       | P1022 | R3              |
| E1       | P1025 | R4              |
+----------+-------+-----------------+
4 rows in set (0.00 sec)
```

TASK 3 [FUNCTIONS]:

1. Create a **function** GetEventCost that accepts an event_id and returns the **total price** a participant would pay to register for that event (i.e., return the event's price from the event table).
   (hint: Simple SELECT with RETURN.)

Before: --describe the table/s

```
mysql> SELECT event_id, event_name, price FROM event WHERE event_id='E1';
+----------+----------------+--------+
| event_id | event_name     | price  |
+----------+----------------+--------+
| E1       | Adventure Trek | 200.00 |
+----------+----------------+--------+
1 row in set (0.00 sec)
```

Command – sql command

```
mysql> DELIMITER //
mysql> CREATE FUNCTION GetEventCost(input_event_id VARCHAR(5))
    -> RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE event_cost DECIMAL(10,2);
    ->     SELECT price INTO event_cost
    ->     FROM event
    ->     WHERE event_id = input_event_id;
    ->     RETURN event_cost;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql>
```

After: --describe the table/s

```
mysql> -- Test
mysql> SELECT GetEventCost('E1');
+--------------------+
| GetEventCost('E1') |
+--------------------+
|             200.00 |
+--------------------+
1 row in set (0.00 sec)
```

2. Create a **function** `GetParticipantPurchaseTotal(SRN)` that returns the **total amount** a participant has spent across all stalls.
   (hint: Aggregate using `SUM(price_per_unit * quantity)` by joining `purchased` and `stall_items`.)

Before: --describe the table/s

```
mysql> SELECT * FROM purchased WHERE SRN='P1001';
+-------+----------+---------------------+---------------------+----------+
| SRN   | stall_id | item_name           | timestamp           | quantity |
+-------+----------+---------------------+---------------------+----------+
| P1001 | S1       | Mushroom Risotto    | 2023-04-15 13:00:05 |        3 |
| P1001 | S1       | Veggie Wrap         | 2023-04-15 12:00:00 |        2 |
| P1001 | S5       | Classic Caesar Salad | 2022-04-17 10:10:00 |        2 |
| P1001 | S5       | Veggie Wrap         | 2022-04-17 10:10:00 |        3 |
+-------+----------+---------------------+---------------------+----------+
4 rows in set (0.00 sec)
```

Command – sql command

```
mysql> SELECT * FROM purchased WHERE SRN='P1001';
+-------+----------+---------------------+---------------------+----------+
| SRN   | stall_id | item_name           | timestamp           | quantity |
+-------+----------+---------------------+---------------------+----------+
| P1001 | S1       | Mushroom Risotto    | 2023-04-15 13:00:05 |        3 |
| P1001 | S1       | Veggie Wrap         | 2023-04-15 12:00:00 |        2 |
| P1001 | S5       | Classic Caesar Salad | 2022-04-17 10:10:00 |        2 |
| P1001 | S5       | Veggie Wrap         | 2022-04-17 10:10:00 |        3 |
+-------+----------+---------------------+---------------------+----------+
4 rows in set (0.00 sec)

mysql> DELIMITER //
mysql> CREATE FUNCTION GetParticipantPurchaseTotal(input_SRN VARCHAR(10))
    -> RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> BEGIN
    ->     DECLARE total_spent DECIMAL(10,2);
    ->     SELECT SUM(si.price_per_unit * p.quantity) INTO total_spent
    ->     FROM purchased p
    ->     JOIN stall_items si
    ->       ON p.stall_id = si.stall_id AND p.item_name = si.item_name
    ->     WHERE p.SRN = input_SRN;
    ->     RETURN IFNULL(total_spent,0.00);
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;
mysql>
```

After: --describe the table/s

```
mysql> -- Test
mysql> SELECT GetParticipantPurchaseTotal('P1001');
+--------------------------------------+
| GetParticipantPurchaseTotal('P1001') |
+--------------------------------------+
|                              3773.50 |
+--------------------------------------+
1 row in set (0.00 sec)
```