Lab 8

Triggers

A trigger is a statement that the system executes automatically as a side effect of modification of database.

To design a trigger mechanism, we must meet two requirements.

- ✓ Specify when a trigger is to be executed. This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.
- ✓ Specify the actions to be taken when the trigger executes

Once we enter a trigger into a database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition satisfied.

Syntax for creating triggers:

```
CREATE [OR REPLACE] TRIGGER trigger_name

{BEFORE | AFTER } {INSERT | UPDATE | DELETE}

ON table_name

[FOR EACH ROW]

[WHEN (condition)]

BEGIN

   -- Trigger body: SQL statements or code

END;
```

CREATE [OR REPLACE] TRIGGER: Specifies the creation of a new trigger or replaces an existing trigger with the same name.

BEFORE | AFTER: Specifies when the trigger should be executed relative to the triggering event. A "BEFORE" trigger fires before the event, and an "AFTER" trigger fires after the event.

INSERT | UPDATE | DELETE: Indicates the event that triggers the execution of the trigger. You can specify multiple events using commas.

ON table_name: Specifies the name of the table on which the trigger is defined.

FOR EACH ROW: Indicates that the trigger should be executed once for each affected row (used for row-level triggers).

BEGIN and END: Enclose the trigger body, which contains the SQL statements or code to be executed when the trigger is fired.

Need of triggers

Triggers can be used to implement certain integrity constraints that cannot be specified using the constraint mechanism of SQL. Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met.

Suppose a warehouse wishes to maintain the a minimum inventory of each item. When inventory level of an items falls below the minimum level, an order can by placed automatically. On update of the inventory level of each item, the trigger compares the current inventory level with minimum inventory level for the item, and if the level is at below new order is created.

Implementation

As we already discussed, the warehouse manages an inventory of various items. Each item has a specific inventory level and a minimum level that should be maintained. Whenever the inventory level falls below the minimum level for an item, the warehouse needs to automatically generate an order to maintain the stock.

Step 1: Create the "Items" table:

First, we need to create a table called "Items" that will store the item details, including the inventory level and the minimum level. The table schema might look like this:

```
CREATE TABLE Items (

   ItemID INT PRIMARY KEY,

   ItemName VARCHAR(50),

   InventoryLevel INT,

   MinimumLevel INT

);
```

Step 2: Insert sample data:

Let's insert some sample data into the "Items" table to represent the initial inventory levels and minimum levels of different items.

```
INSERT INTO Items (ItemID, ItemName, InventoryLevel, MinimumLevel)

VALUES (1, 'Item A', 10, 5),

       (2, 'Item B', 8, 6),

       (3, 'Item C', 15, 10);
```

Step 3: Create the "Orders" table:

We need a table to store the generated orders when the inventory level falls below the minimum level. Create a table called "Orders" with the necessary columns to store the order details.

```
CREATE TABLE Orders (

   OrderID INT AUTO_INCREMENT PRIMARY KEY,

   ItemID INT,

   Quantity INT,

   OrderDate DATE

);
```

Step 4: Create the trigger:

Now, we'll create a trigger that will automatically place an order when the inventory level falls below the minimum level for any item. The trigger will be executed after an update operation on the "Items" table.

```
DELIMITER //

CREATE TRIGGER check_inventory_level

AFTER UPDATE ON Items

FOR EACH ROW

BEGIN

  DECLARE current_inventory INT;

  DECLARE minimum_level INT;

  SET current_inventory = NEW.InventoryLevel;

  SET minimum_level = NEW.MinimumLevel;


  IF current_inventory < minimum_level THEN

    INSERT INTO Orders (ItemID, Quantity, OrderDate)

    VALUES (NEW.ItemID, (minimum_level - current_inventory), CURDATE());

  END IF;

END //

DELIMITER ;
```

In this trigger:

- The trigger is defined as an "AFTER UPDATE" trigger on the "Items" table.
- For each updated row, the trigger checks the current inventory level (NEW.InventoryLevel) and the minimum inventory level (NEW.MinimumLevel).
- If the current inventory level is less than minimum level, a new order is inserted into the "Orders" table. The order includes the ItemID, the quantity needed to reach the minimum level (minimum_level - current_inventory), and the current date (CURDATE()).

Step 5: Update inventory levels:

Now, let's update the inventory levels of the items to simulate the scenario where some of the items fall below their minimum levels.

```
UPDATE Items SET InventoryLevel = 3 WHERE ItemID = 1;

-- Item A falls below minimum level

UPDATE Items SET InventoryLevel = 4 WHERE ItemID = 2;

 -- Item B remains above minimum level
```

Step 6: Check the "Orders" table:

After executing the update statements, you can check the "Orders" table to see if any orders have been automatically generated due to the trigger.

```
SELECT * FROM Orders;
```