

## **LAB NO 6**

### **TRIGGERS AND SUBQUERIES IN SQL**

#### **Objective:**

Study Triggers and Types of Triggers. Also Apply sub queries and correlated subqueries in SQL.

#### **LAB TASKS**

1. Create an insert trigger for the products table. Whenever products are inserted in the table 'products', the table 'products\_log' with the columns (product id, pro\_quantity, date, action) shall be updated. Use the getdate() function to insert the current date for the products\_log table.

#### **Input 01:**

```
30 • create table Products      (product_id int,  pro_quantity int , pro_date date);
31 • create table Products_log (product_id int,  pro_quantity int , pro_date date, action varchar(25) );
32
33 • CREATE TRIGGER insert_products_notifier
34   AFTER INSERT ON Products
35   FOR EACH ROW
36
37       INSERT INTO Products_log (product_id, pro_quantity, pro_date, action)
38
39   VALUES (NEW.product_id, NEW.pro_quantity, NOW(), 'INSERT');
40
41 • insert into Products (product_id, pro_quantity, pro_date)
42   values
43   (0021 , 50 , '2025-05-17'),
44   (0022 , 70 , '2025-05-19'),
45   (0023 , 40 , '2025-05-15'),
46   (0024 , 10 , '2025-05-14'),
47   (0025 , 90 , '2025-05-11');
48
49 • select * from products_Log;
```

#### **Output 01:**

```

49 • select * from products_Log;
50

```

Result Grid | Filter Rows: | Exports:

	product_id	pro_quantity	pro_date	action
▶	21	50	2025-05-17	INSERT
	22	70	2025-05-17	INSERT
	23	40	2025-05-17	INSERT
	24	10	2025-05-17	INSERT
	25	90	2025-05-17	INSERT

2. Create a delete trigger for the products table with the columns (Product\_id, product\_name, quantity, unit price).

## Input 02:

```

Limit to 1000 rows
56 • create table Products (product_id int, pro_quantity int, pro_date date);
57 • create table Products_log (product_id int, pro_quantity int, pro_date date, action varchar(25) );
58
59 • CREATE TRIGGER delete_products_notifier
60 AFTER DELETE ON Products
61 FOR EACH ROW
62
63 INSERT INTO Products_log (product_id, pro_quantity, pro_date, action)
64
65 VALUES (old.product_id, old.pro_quantity, NOW(), 'DELETE');
66
67 • insert into Products (product_id, pro_quantity, pro_date)
68 values
69 (0021 , 50 , '2025-05-17'),
70 (0022 , 70 , '2025-05-19'),
71 (0023 , 40 , '2025-05-15'),
72 (0024 , 10 , '2025-05-14'),
73 (0025 , 90 , '2025-05-11');
74
75 • select * from products;
76 • SET SQL_SAFE_UPDATES = 0;
77 • delete from Products where product_id = 21;
78
79 • select * from Products_log;

```

## Output 02:

	product_id	pro_quantity	pro_date	action
	21	50	2025-05-17	INSERT
	22	70	2025-05-17	INSERT
	23	40	2025-05-17	INSERT
	24	10	2025-05-17	INSERT
	25	90	2025-05-17	INSERT
▶	21	50	2025-05-17	DELETE

3. Create an instead of trigger which restricts the user to delete the product whose product id is 5.

### Input 03:

```
--3
-- Step 1: Create the INSTEAD OF DELETE trigger
CREATE TRIGGER trg_instead_of_delete_product5
ON products
INSTEAD OF DELETE
AS
BEGIN
    -- Check if the deleted row includes product_id = 5
    IF EXISTS (SELECT * FROM deleted WHERE product_id = 5)
    BEGIN
        -- Prevent deletion and show error
        RAISERROR ('Deletion of product with ID = 5 is not allowed.', 16, 1);
    END
    ELSE
    BEGIN
        -- Allow deletion for other products
        DELETE FROM products
        WHERE product_id IN (SELECT product_id FROM deleted);
    END
END;
-- Step 2: Try to delete product with ID 5 (this should be blocked)
DELETE FROM products WHERE product_id = 5;
-- Step 3: Try to delete product with another ID (this will be allowed)
DELETE FROM products WHERE product_id = 1;
```

### Output 03:

100 %

Messages

Msg 50000, Level 16, State 1, Procedure trg\_instead\_of\_delete\_product5, Line 10 [Batch Start Line 102]  
Deletion of product with ID = 5 is not allowed.

(1 row affected)

Completion time: 2025-05-18T19:13:17.4599099+05:00

4. Create an update trigger which triggers when there is any update in the product quantity or product name or product unit price. Maintain the log in the products\_log table which has been mentioned above.

## Input 04:

```
90 • CREATE TRIGGER update_products_notifier
91     AFTER UPDATE ON Products
92     FOR EACH ROW
93
94     INSERT INTO Updates_notifier (product_id, pro_quantity, pro_date, action)
95     VALUES (new.product_id, new.pro_quantity, NOW(), 'UPDATE');
96
97 • insert into Products (product_id, pro_quantity, pro_date)
98     values
99     (0021 , 50 , '2025-05-17'),
100     (0022 , 70 , '2025-05-19'),
101     (0023 , 40 , '2025-05-15'),
102     (0024 , 10 , '2025-05-14'),
103     (0025 , 90 , '2025-05-11');
104
105 • select * from products;
106
107 • update Products
108     set product_id = 99 where product_id =22 ;
109 • select * from Updates_notifier;
110
```

## Output 04:

```
109 • select * from Updates_notifier;
110
```

product_id	pro_quantity	pro_date	action
99	70	2025-05-17	UPDATE

5. Explore the IN, EXISTS, ANY , NOT EXISTS clause and embed them in your query . Also explain their use cases.

## IN:

```
117 • select * from products where product_id IN (23,24,25) ;
118 -- Iska matlab sirf un rows ko dikhao jinki product_id 23, 24, ya 25 hai.
```

	product_id	pro_quantity	pro_date
▶	23	40	2025-05-15
	24	10	2025-05-14
	25	90	2025-05-11

## EXISTS:

```
120 • SELECT * FROM Products p -- Products table ke saare rows uthao or ose p se refer kro.
121 WHERE EXISTS ( -- Ye check karta hai: Kya andar wali subquery kuch bhi data return karti hai
122
123     SELECT 1 FROM Products_log pl
124     -- Yahan 1 ka matlab kuch bhi value ho sakti hai – EXISTS sirf yeh dekhta hai ke kuch aaya ya nahi.
125     -- To phir SELECT 1 kyu likhte hain?
126     -- Kyun ke EXISTS ko sirf existence check karni hoti hai, actual data kaam ka nahi hota.
127     -- 1 ek dummy value hai, jo batata hai: "haan, row mil gaya!"
128
129     WHERE pl.product_id = p.product_id);
```

	product_id	pro_quantity	pro_date
▶	23	40	2025-05-15
	24	10	2025-05-14
	25	90	2025-05-11

## Not EXISTS:

```
133 • SELECT * FROM Products p
134 WHERE Not EXISTS (
135     SELECT 1 FROM Products_log pl
136     WHERE pl.product_id = p.product_id
137 );
```

	product_id	pro_quantity	pro_date
▶	99	70	2025-05-19

## Any:

```

11 • SELECT * FROM Products
12 WHERE pro_quantity > ANY (
13     SELECT pro_quantity FROM Products_log
14 );

```

	product_id	pro_quantity	pro_date
▶	99	70	2025-05-19
	23	40	2025-05-15
	25	90	2025-05-11

6. Write a query to list all customers (CustomerID, FirstName, LastName) who have placed orders. Use a correlated subquery with EXISTS to check if a customer has any orders in the Orders table.

## Input 06:

```

145
146 • create table customer(C_id int , C_Firstname varchar(25) ,C_Lastname varchar(25) , C_order_id int );
147 • insert into customer (C_id , C_Firstname ,C_Lastname ,C_order_id ) values
148     (221, 'Waqar', 'Riasat',001),
149     (216, 'Hussain', 'Raza',002),
150     (219, 'Qasim' , 'Qadri',003),
151     (230, 'Aneeq' , 'Shms',004),
152     (250, 'Saqib' , 'Siddique',005);
153
154 • create table Order_id (order_id int , order_type varchar(25));
155 • insert into Order_id(order_id , order_type)values
156     (001,'Confirmed'),
157     (002,'Confirmed'),
158     (005,'Confirmed');
159
160
161 • select * from customer c
162 where exists (
163     select 1 from Order_id o
164     where c.C_order_id = o.order_id );

```

## Output 06:

	C_id	C_Firstname	C_Lastname	C_order_id
▶	221	Waqar	Riasat	1
	216	Hussain	Raza	2
	250	Saqib	Siddique	5

7. Create a function which displays the ids and names of those students who have got gpa greater than or equal to three

## Input 07:

```
169 • create table student (std_id int , std_name varchar(25), gpa double) ;
170 • insert into student (std_id , std_name , gpa) values
171     (221, 'waqar', 3.2),
172     (250,'saqib',3.85),
173     (230,'aneeq', 3.75),
174     (216,'hussain' , 2.95),
175     (222, 'shntu' , 1.4),
176     (227, 'Umer' , 1.79);
177
178 • create procedure show_scholarship_students()
179     select std_id ,std_name , gpa
180     from student
181     where gpa >=3.0 ;
182
183 • call show_scholarship_students();
```

## Output 07:

	std_id	std_name	gpa
▶	221	waqar	3.2
	250	saqib	3.85
	230	aneeq	3.75

### why we used a stored procedure instead of a function:

- SQL functions must return a single value, like a number, string, or boolean — not a result set.
- You can't write RETURN SELECT \* FROM ... in a function.

Answer the following questions:

#### • What have you learned from the lab task?

I learned how triggers work in databases and their different types—BEFORE, AFTER, and INSTEAD OF triggers. I understood how triggers automatically execute in response to data changes like INSERT, UPDATE, or DELETE.

- **What was the most challenging task and how did you overcome that challenge?**

The most challenging task for me was writing triggers correctly and modify data without causing errors . I overcame this by carefully testing each trigger step-by-step and using conditions to control when they fire.