Day 9 Assignment

PostgreSQL

1. Create AFTER UPDATE trigger to track product price changes

a. Create product_price_audit table with below columns:

Query | Query History

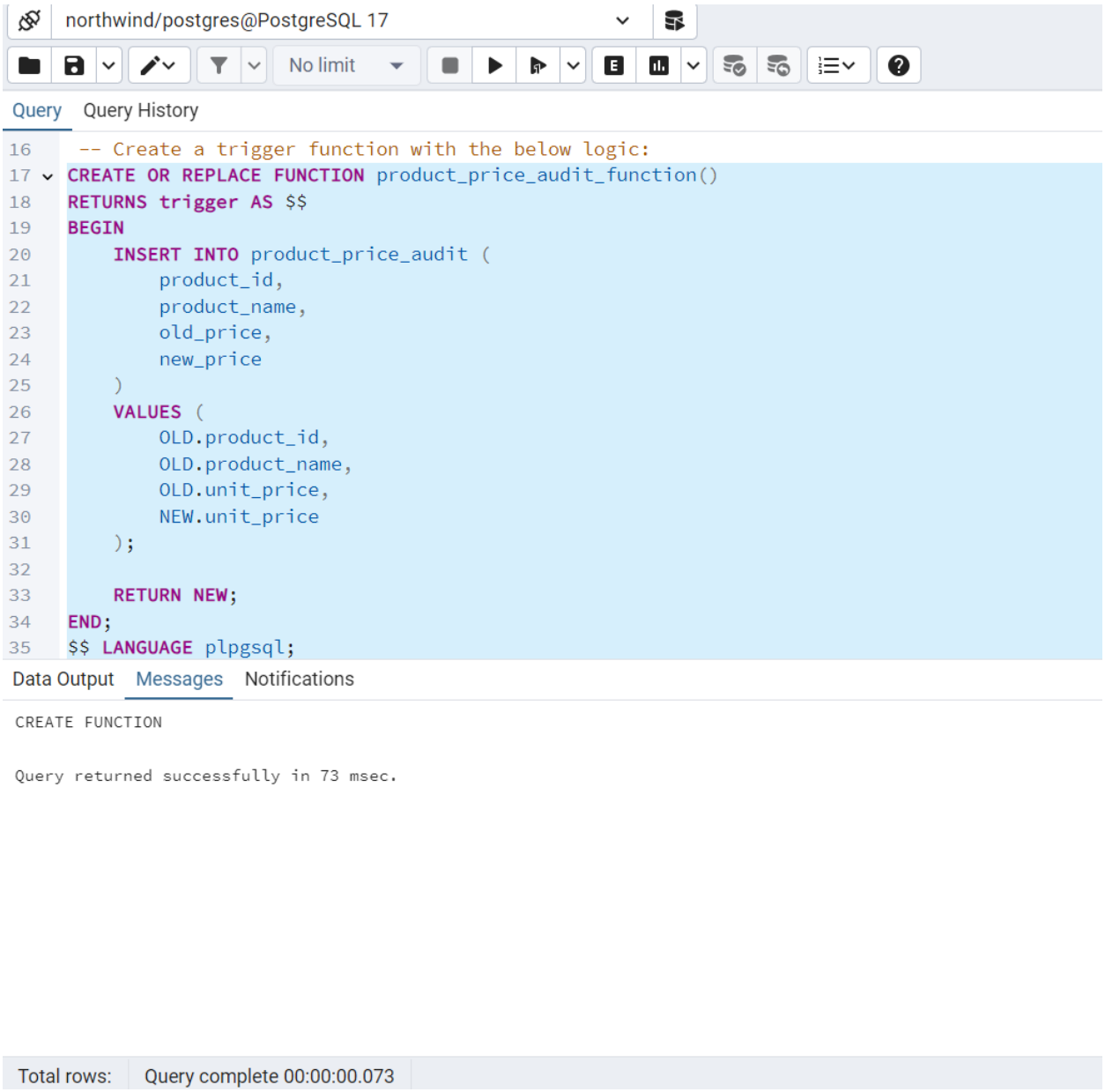
```
1
2   -- 1. Create AFTER UPDATE trigger to track product price changes
3
4   -- Create product_price_audit table
5
6   CREATE TABLE product_price_audit (
7       audit_id SERIAL PRIMARY KEY,
8       product_id INT,
9       product_name VARCHAR(40),
10      old_price DECIMAL(10,2),
11      new_price DECIMAL(10,2),
12      change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
13      user_name VARCHAR(50) DEFAULT CURRENT_USER
14  );
15
16   -- Create a trigger function with the below logic:
17
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 87 msec.
```

b. Create a trigger function with the below logic:



c. Create a row level trigger for below event:

```sql
-- Create a row level trigger for below event:
CREATE TRIGGER product_price_audit_trigger
AFTER UPDATE OF unit_price ON products
FOR EACH ROW
WHEN (OLD.unit_price IS DISTINCT FROM NEW.unit_price)
EXECUTE FUNCTION product_price_audit_function();
```

Data Output | Messages | Notifications

```
CREATE TRIGGER

Query returned successfully in 60 msec.
```

Total rows: | Query complete 00:00:00.060

d. Test the trigger by updating the product price by 10% to any one product_id.

Before update

```
43
44
45  select * from products WHERE product_id = 1 ;
46  select * from product_price_audit;
47
48
```

Data Output | Messages | Notifications

	audit_id [PK] integer	product_id integer	product_name character varying (40)	old_price numeric (10,2)	new_price numeric (10,2)	change_date timestamp without time zone	user_name character varying (50)

Total rows: 0 | Query complete 00:00:00.135

After update:

```
48  -- Test the trigger by updating the product price by 10% to any one product_id.
49
50  UPDATE products
51  SET unit_price = unit_price * 1.10
52  WHERE product_id = 1;
53
54  select * from products WHERE product_id = 1 ;
55  select * from product_price_audit;
56
57
58  ·       Test the trigger by updating the product price by 10% to any one product_id.
```

Data Output | Messages | Notifications

Showing

	audit_id [PK] integer	product_id integer	product_name character varying (40)	old_price numeric (10,2)	new_price numeric (10,2)	change_date timestamp without time zone	user_name character varying (50)
1	1	1	Chai	19.80	21.78	2025-05-05 01:35:00.663074	postgres

Total rows: 1 | Query complete 00:00:00.098

2. Create stored procedure using IN and INOUT parameters to assign tasks to employees

a. Create table employee_tasks:

```
58  -- 2 . Create stored procedure using IN and INOUT parameters to assign tasks to employees.*/
59
60  -- Create table employee_tasks
61  CREATE TABLE IF NOT EXISTS employee_tasks (
62          task_id SERIAL PRIMARY KEY,
63          employee_id INT,
64          task_name VARCHAR(50),
65          assigned_date DATE DEFAULT CURRENT_DATE);
66
67
68
69
70
```

Data Output | Messages | Notifications

```
CREATE TABLE

Query returned successfully in 76 msec.
```

Total rows: | Query complete 00:00:00.076

b. Create procedure

Query | Query History

```
68  -- Create a Stored Procedure
69  CREATE OR REPLACE PROCEDURE assign_task (
70  IN p_employee_id INT,
71  IN p_task_name VARCHAR(50),
72  INOUT p_task_count INT DEFAULT 0
73  )
74  LANGUAGE plpgsql
75  AS $$
76  BEGIN
77  -- Insert employee_id, task_name  into employee_tasks
78      INSERT INTO employee_tasks (employee_id, task_name)
79      VALUES (p_employee_id, p_task_name);
80
81  -- Count total tasks for employee and put the total count into p_task_count
82      SELECT COUNT(*) INTO p_task_count
83      FROM employee_tasks
84      WHERE employee_id = p_employee_id;
85
86   -- Raise NOTICE message:
87      RAISE NOTICE 'Task "%" assigned to employee %. Total tasks: %',
88          p_task_name, p_employee_id, p_task_count;
89  END;
90  $$;
```

Data Output | Messages | Notifications

```
CREATE PROCEDURE

Query returned successfully in 110 msec.
```

Total rows: | Query complete 00:00:00.110

c. After creating stored procedure test by calling it:

```
$$;

--> Step 3: Call the Stored Procedure
CALL assign_task(1, 'Review Reports');

--> You should see the entry in employee_tasks table.
SELECT * FROM employee_tasks;
```

Data Output | Messages | Notifications

	p_task_count integer
1	1

Total rows: 1 | Query complete 00:00:00.149

d. You should see the entry in employee_tasks table.

```
-- You should see the entry in employee_tasks table.
SELECT * FROM employee_tasks;
```

Data Output | Messages | Notifications

	task_id [PK] integer	employee_id integer	task_name character varying (50)	assigned_date date
1	1	1	Review Reports	2025-05-05

Total rows: 1 | Query complete 00:00:00.206