

## Day 9 Assignment

### 1. Create AFTER UPDATE trigger to track product price changes

```
CREATE TABLE IF NOT EXISTS product_price_audit (  
    audit_id SERIAL PRIMARY KEY,  
    product_id INT,  
    product_name VARCHAR(40),  
    old_price DECIMAL(10,2),  
    new_price DECIMAL(10,2),  
    change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
    user_name VARCHAR(50) DEFAULT CURRENT_USER  
);
```

```
4 CREATE TABLE IF NOT EXISTS product_price_audit (  
5     audit_id SERIAL PRIMARY KEY,  
6     product_id INT,  
7     product_name VARCHAR(40),  
8     old_price DECIMAL(10,2),  
9     new_price DECIMAL(10,2),  
10    change_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
11    user_name VARCHAR(50) DEFAULT CURRENT_USER  
12 );
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 74 msec.

#### Step 1: Create a trigger function with the below logic.

```
CREATE OR REPLACE FUNCTION fn_track_price_change()
```

```
RETURNS TRIGGER AS $$
```

```
BEGIN
```

```
    INSERT INTO product_price_audit (  
        product_id,  
        product_name,  
        old_price,
```

```

        new_price
    )
VALUES (
    OLD.product_id,
    OLD.product_name,
    OLD.unit_price,
    NEW.unit_price
);
RETURN NEW;
END;
$$ LANGUAGE plpgsql;

```

```

15 CREATE OR REPLACE FUNCTION fn_track_price_change()
16 RETURNS TRIGGER AS $$
17 BEGIN
18     INSERT INTO product_price_audit (
19         product_id,
20         product_name,
21         old_price,
22         new_price
23     )
24     VALUES (
25         OLD.product_id,
26         OLD.product_name,
27         OLD.unit_price,
28         NEW.unit_price
29     );
30     RETURN NEW;
31 END;
32 $$ LANGUAGE plpgsql;

```

Data Output Messages Notifications

CREATE FUNCTION

Query returned successfully in 72 msec.

**Step 3: Create a row level trigger for the event below.**

```
CREATE TRIGGER trg_after_price_update
AFTER UPDATE OF unit_price ON products
FOR EACH ROW
EXECUTE FUNCTION fn_track_price_change();
```

```
35 CREATE TRIGGER trg_after_price_update
36 AFTER UPDATE OF unit_price ON products
37 FOR EACH ROW
38 EXECUTE FUNCTION fn_track_price_change();
```

Data Output Messages Notifications

CREATE TRIGGER

Query returned successfully in 95 msec.

#### Step 4: Test the trigger by updating the product price by 10% to any one

```
UPDATE products
SET unit_price = unit_price * 1.10
WHERE product_id = 1;
```

```
42 UPDATE products
43 SET unit_price = unit_price * 1.10
44 WHERE product_id = 1;
```

Data Output Messages Notifications

UPDATE 1

Query returned successfully in 57 msec.

```
SELECT * FROM product_price_audit ORDER BY change_date DESC;
```

```
48 SELECT * FROM product_price_audit ORDER BY change_date DESC;
```

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)
1	1	1	Chai	18.00	19.80	2025-05-05 18:13:25.185912	postgres

## 2. Stored Procedure to Assign Tasks to Employees

### Step 1: Create the employee\_tasks table

```
CREATE TABLE IF NOT EXISTS employee_tasks (  
    task_id SERIAL PRIMARY KEY,  
    employee_id INT,  
    task_name VARCHAR(50),  
    assigned_date DATE DEFAULT CURRENT_DATE  
);
```

```
53 CREATE TABLE IF NOT EXISTS employee_tasks (  
54     task_id SERIAL PRIMARY KEY,  
55     employee_id INT,  
56     task_name VARCHAR(50),  
57     assigned_date DATE DEFAULT CURRENT_DATE  
58 );
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 99 msec.

### Step 2: Create the stored procedure

```
CREATE OR REPLACE PROCEDURE assign_task(  
    IN p_employee_id INT,  
    IN p_task_name VARCHAR(50),  
    INOUT p_task_count INT DEFAULT 0  
)  
LANGUAGE plpgsql  
AS $$  
BEGIN  
    -- Insert new task  
    INSERT INTO employee_tasks (employee_id, task_name)  
    VALUES (p_employee_id, p_task_name);
```

```
-- Count total tasks for the employee
SELECT COUNT(*) INTO p_task_count
FROM employee_tasks
WHERE employee_id = p_employee_id;
```

```
-- Output message
```

```
RAISE NOTICE 'Task "%" assigned to employee %. Total tasks: %',
    p_task_name, p_employee_id, p_task_count;
```

```
END;
```

```
$$;
```

```
61 ✓ CREATE OR REPLACE PROCEDURE assign_task(
62     IN p_employee_id INT,
63     IN p_task_name VARCHAR(50),
64     INOUT p_task_count INT DEFAULT 0
65 )
66 LANGUAGE plpgsql
67 AS $$
68 BEGIN
69     -- Insert new task
70     INSERT INTO employee_tasks (employee_id, task_name)
71     VALUES (p_employee_id, p_task_name);
72
73     -- Count total tasks for the employee
74 ✓ SELECT COUNT(*) INTO p_task_count
75     FROM employee_tasks
76     WHERE employee_id = p_employee_id;
77
78     -- Output message
79 ✓ RAISE NOTICE 'Task "%" assigned to employee %. Total tasks: %',
80         p_task_name, p_employee_id, p_task_count;
81 END;
82 $$;
```

Data Output Messages Notifications

CREATE PROCEDURE

Query returned successfully in 67 msec.

### Step 3: Call the procedure and test

CALL assign\_task(1, 'Review Reports');

```
84  
85 CALL assign_task(1, 'Review Reports');  
86 -- Check the result:
```

Data Output Messages Notifications

	p_task_count integer
1	1

### Check the result:

SELECT \* FROM employee\_tasks WHERE employee\_id = 1;

```
89 SELECT * FROM employee_tasks WHERE employee_id = 1;  
90  
91  
92  
93  
94
```

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