**Day 9**

1. Create AFTER UPDATE trigger to track product price changes

· Create product\_price\_audit table with below columns:

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

· Create a trigger function with the below logic:

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

);

· Create a row level trigger for below event:

AFTER UPDATE OF unit\_price ON products

· Test the trigger by updating the product price by 10% to any one product\_id.

Here , creating a Product\_Price table —

CREATE TABLE product\_price\_audit table

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

);

In the second step we are creating Trigger-

CREATE OR REPLACE Trigger track\_price\_change()

RETURNS TRIGGER AS $$

BEGIN

-- Only log if price actually changed

IF OLD.unit\_price IS DISTINCT FROM NEW.unit\_price THEN

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

);

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

Updating Trigger here

CREATE TRIGGER trg\_track\_price\_change

AFTER UPDATE OF unit\_price ON products

FOR EACH ROW

EXECUTE FUNCTION track\_price\_change();

Checking updating value via this query

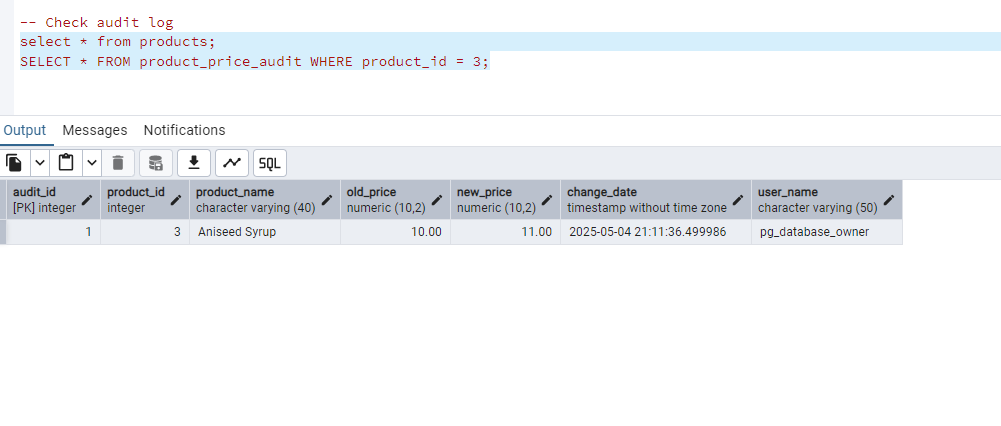
SELECT product\_id, product\_name, unit\_price FROM products WHERE product\_id = 3;

-- Update price by 10%

UPDATE products

SET unit\_price = unit\_price \* 1.10

WHERE product\_id = 3;

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

· Parameters:

IN p\_employee\_id INT,

IN p\_task\_name VARCHAR(50),

INOUT p\_task\_count INT DEFAULT 0

· Inside Logic: Create table employee\_tasks:

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

);

· Insert employee\_id, task\_name into employee\_tasks

· Count total tasks for employee and put the total count into p\_task\_count .

· Raise NOTICE message:  
 RAISE NOTICE 'Task "%" assigned to employee %. Total tasks: %',

p\_task\_name, p\_employee\_id, p\_task\_count;

After creating stored procedure test by calling it:

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

You should see the entry in employee\_tasks table.

**Create assign tasks to employees 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

);

**Create Procedure here-**

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 the task**

INSERT INTO employee\_tasks (employee\_id, task\_name)

VALUES (p\_employee\_id, p\_task\_name);

**-Count total tasks assigned to the employee**

SELECT COUNT(\*) INTO p\_task\_count

FROM employee\_tasks

WHERE employee\_id = p\_employee\_id;

DO $$

DECLARE

v\_task\_count INT := 0;

BEGIN

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

-- Optionally display the final count

RAISE NOTICE 'Returned task count: %', v\_task\_count;

END;

$$;

**Verify the employee table -**

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

**Output Verified -**