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,

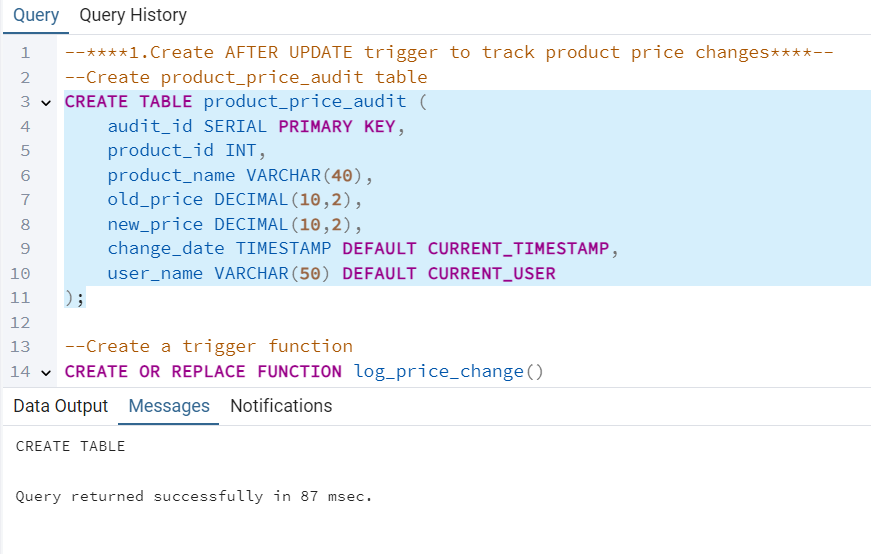
    product\_name VARCHAR(40),

    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,

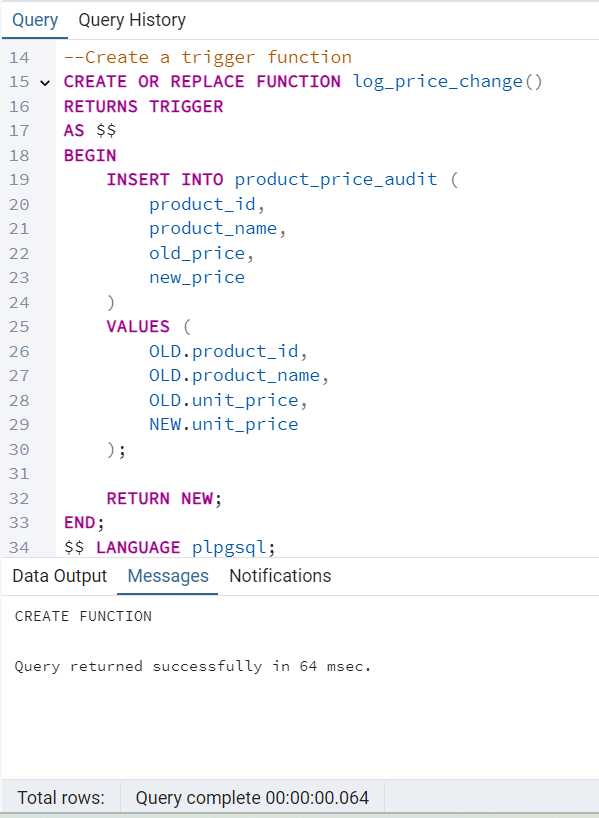
 old\_price,

 new\_price  )

    VALUES (  
OLD.product\_id,  
OLD.product\_name,        OLD.unit\_price,

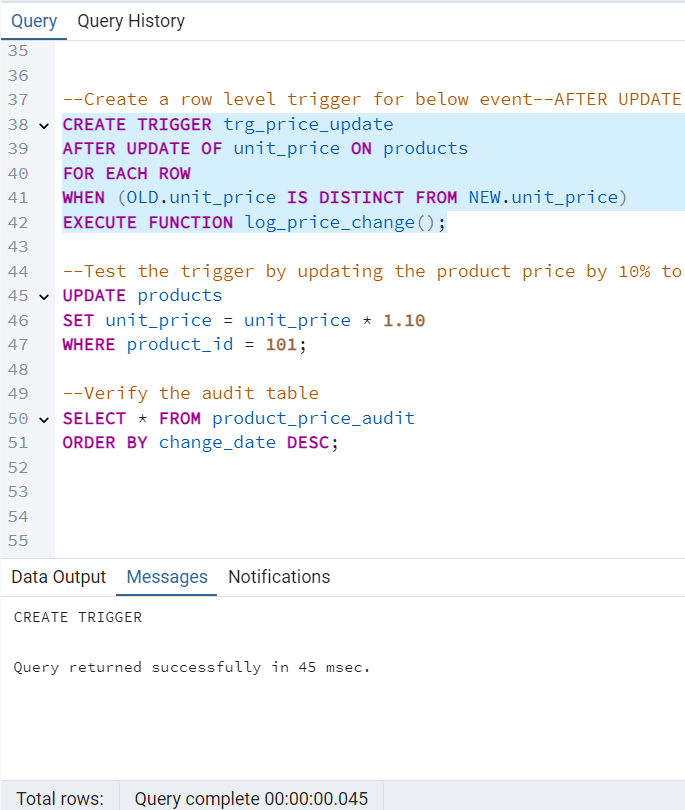
        NEW.unit\_price

    );

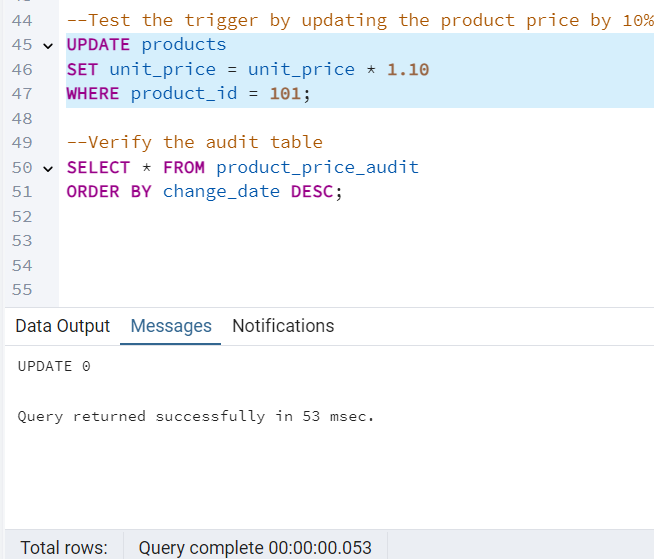


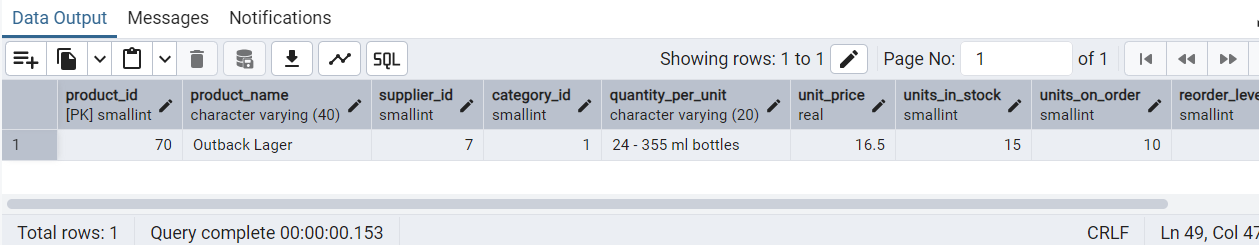
Create a row level trigger for below event:

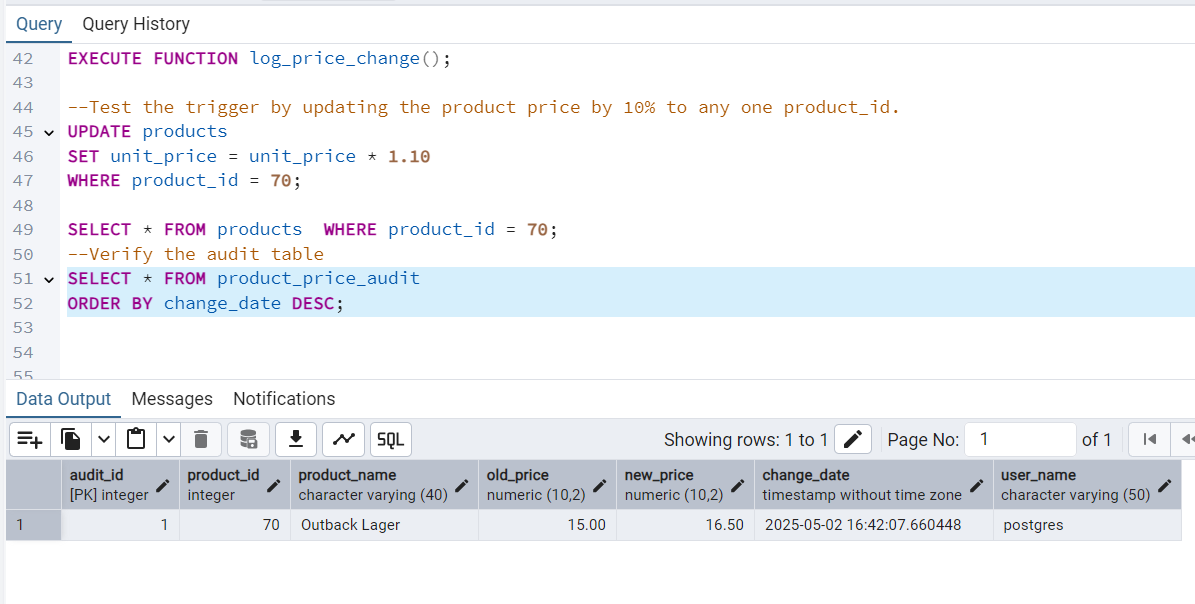
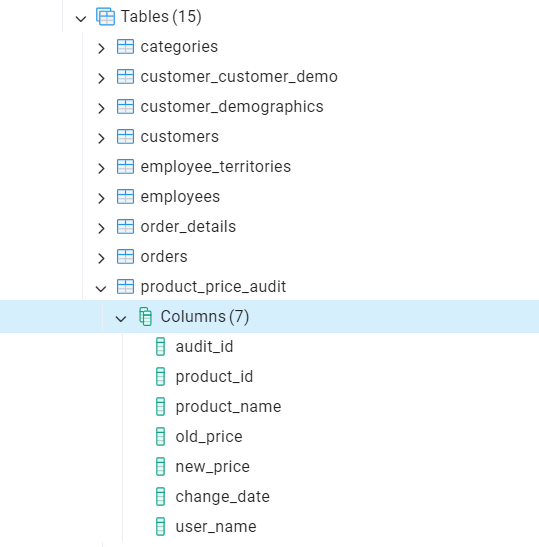
           AFTER UPDATE OF unit\_price ON products



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



 Verifying the audit\_table:

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;

