**DAY 9 ASSIGNMENT SQL**

--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.

– 1 We first create the table :

create table product\_price\_audit(

audit\_id SERIAL PRIMARY KEY,

product\_id INT,

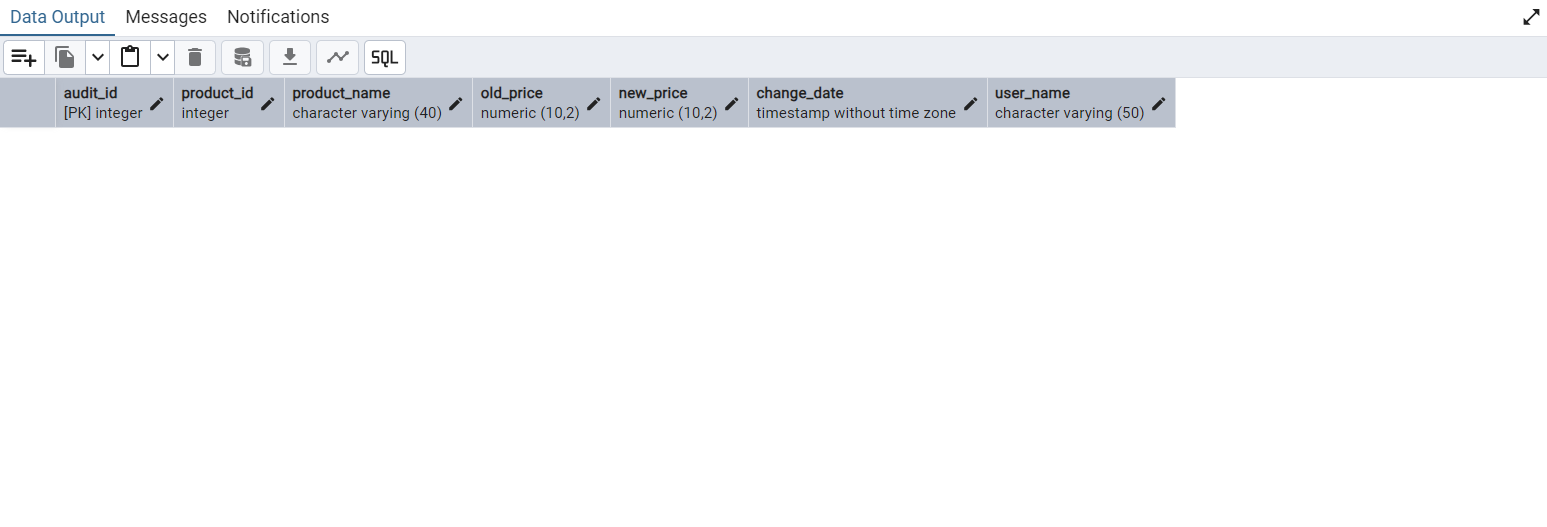
product\_name VARCHAR(40),

old\_price DECIMAL(10,2),

new\_price DECIMAL(10,2),

change\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

user\_name VARCHAR(50) DEFAULT CURRENT\_USER)



CREATE OR REPLACE FUNCTION log\_new\_price()

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;

CREATE TRIGGER product\_update

AFTER UPDATE OF unit\_price ON products

FOR EACH ROW

WHEN (OLD.unit\_price IS DISTINCT FROM NEW.unit\_price)

EXECUTE FUNCTION log\_new\_price();

--test the TRIGGER

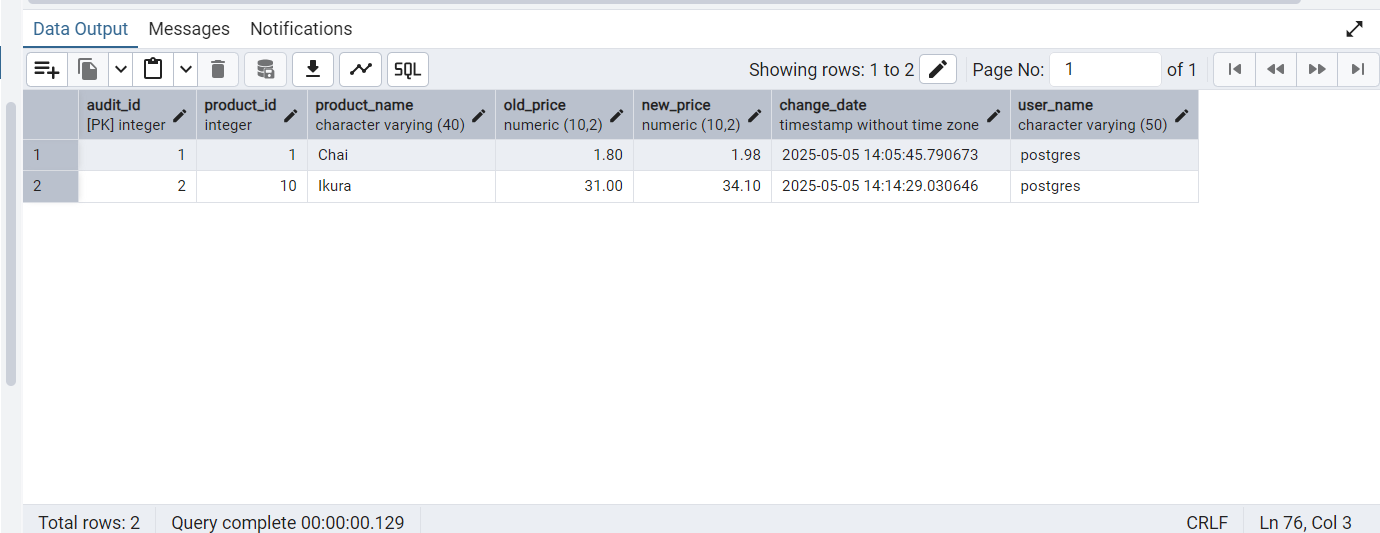
UPDATE products

SET unit\_price = unit\_price \* 1.10

WHERE product\_id = 10;

SELECT \* FROM products

select \* from product\_price\_audit



========================================================================

-- 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

--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');

CREATE TABLE IF NOT EXISTS employee\_tasks(

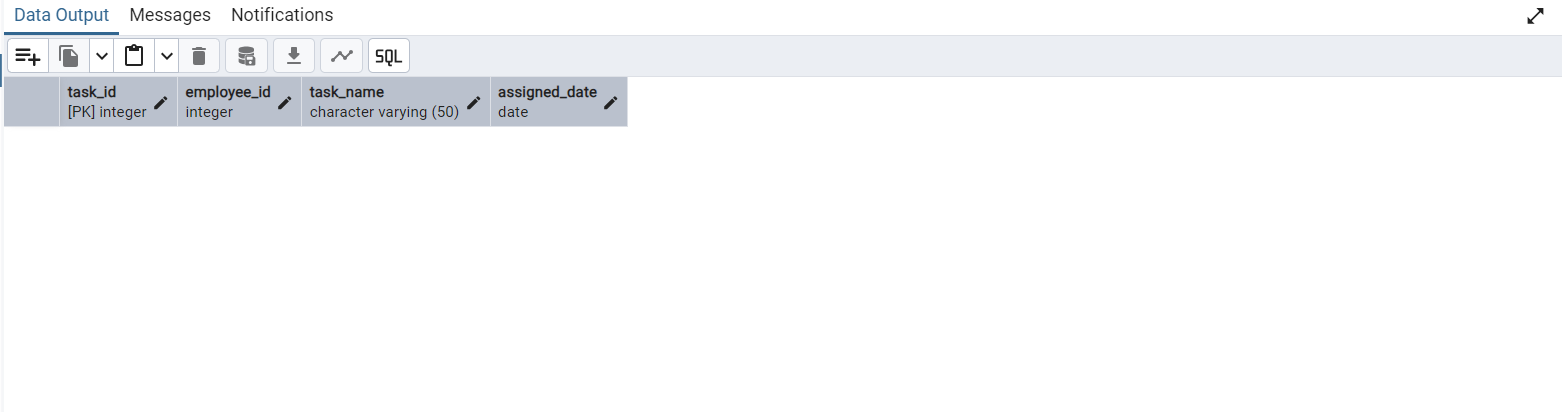
task\_id SERIAL PRIMARY KEY,

employee\_id INT,

task\_name VARCHAR(50),

assigned\_date DATE DEFAULT CURRENT\_DATE

)



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

VALUES(1,'Insert Data'),

(2,'Update Data'),

(3,'Delete Data')

SELECT \* FROM employee\_tasks

CREATE OR REPLACE procedure assign\_employeetask\_value(

IN p\_employee\_id INT,

IN p\_task\_name VARCHAR(50),

INOUT p\_task\_count INT DEFAULT 0

)

LANGUAGE plpgsql

AS $$

BEGIN

IF NOT EXISTS (SELECT 1 FROM employee\_tasks WHERE employee\_id=p\_employee\_id ) THEN

RAISE EXCEPTION 'Employee\_id % doesnot exist',p\_employee\_id;

END IF;

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

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

SELECT COUNT(task\_id)INTO p\_task\_count

FROM employee\_tasks

WHERE employee\_id=p\_employee\_id;

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

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

END;

$$;

CALL assign\_employeetask\_value(1,'delete Reports');

**Stored procedure created:**

