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

CREATE TABLE PRODUCT\_PRICE\_AUDIT (

AUDIT\_ID SERIAL PRIMARY KEY,

PRODUCT\_ID INT,

PRODUCT\_NAME VARCHAR (40),

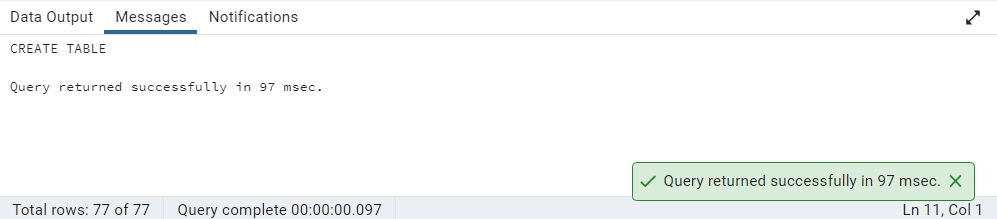
OLD\_PRICE DECIMAL (10,2),

NEW\_PRICE DECIMAL (10,2),

CHANGE\_DATE TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

USER\_NAME VARCHAR (50) DEFAULT CURRENT\_USER

)



--Define the Trigger Function

CREATE OR REPLACE FUNCTION AFTER\_UPDATE\_PRODUCT() 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,

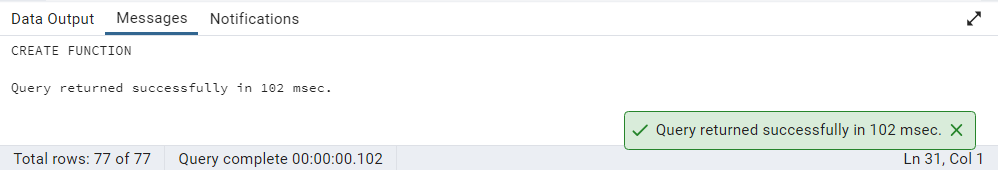
NEW.unit\_price

);

RETURN NEW;

END;

$$ LANGUAGE PLPGSQL

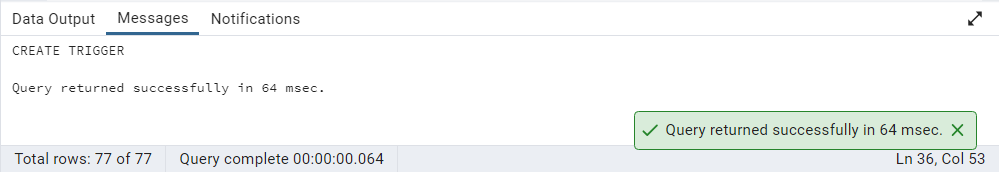


--Create the Trigger

CREATE TRIGGER AFTER\_UPDATE\_TRIGGER AFTER

UPDATE ON PRODUCTS

FOR EACH ROW EXECUTE FUNCTION AFTER\_UPDATE\_PRODUCT()

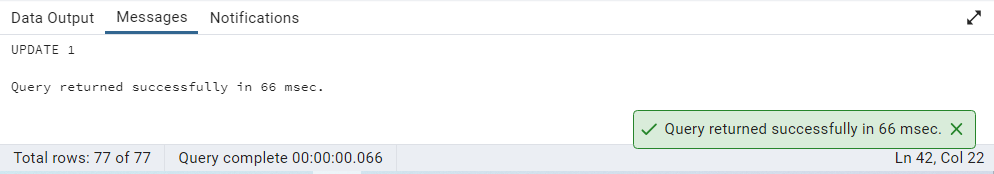


--Test the trigger

UPDATE PRODUCTS

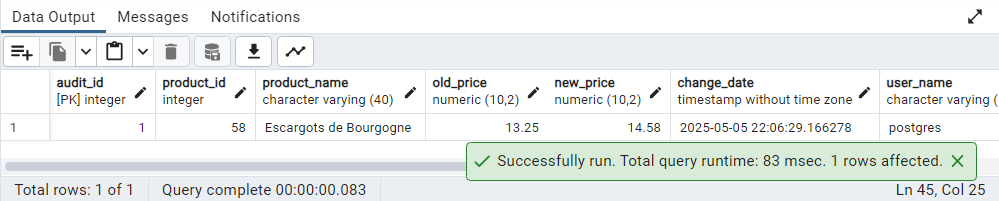
SET UNIT\_PRICE = UNIT\_PRICE \* 1.10

WHERE PRODUCT\_ID = 58



SELECT \*

FROM PRODUCT\_PRICE\_AUDIT



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

CREATE TABLE IF NOT EXISTS EMPLOYEE\_TASKS (

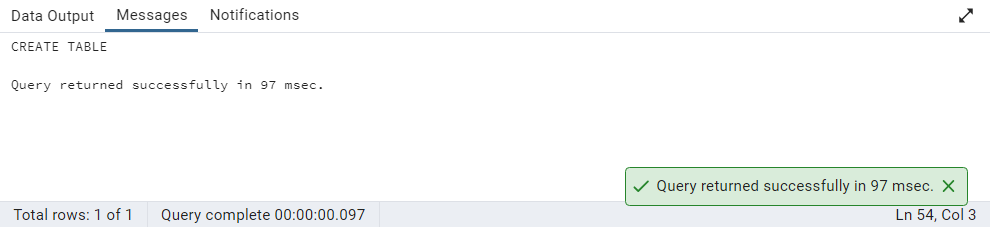
TASK\_ID SERIAL PRIMARY KEY,

EMPLOYEE\_ID INT,

TASK\_NAME VARCHAR(50),

ASSIGNED\_DATE DATE DEFAULT CURRENT\_DATE

);



--Create 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 INTO EMPLOYEE\_TASKS (

EMPLOYEE\_ID, TASK\_NAME

) VALUES (

p\_employee\_id, p\_task\_name

);

SELECT COUNT (\*)

INTO p\_task\_count

FROM EMPLOYEE\_TASKS

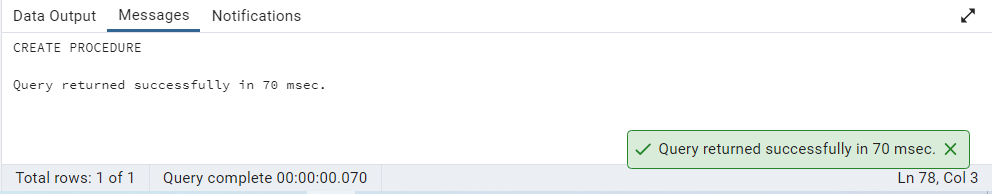
WHERE EMPLOYEE\_ID = p\_employee\_id;

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

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

END;

$$



CALL ASSIGN\_TASK(

1,'Review Reports'

);

