**--Q1) Create view vw\_updatable\_products (use same query whatever I used in the training)**

**--Try updating view with below query and see if the product table also gets updated.**

**--Update query:**

**--UPDATE updatable\_products SET unit\_price = unit\_price \* 1.1 WHERE units\_in\_stock < 10;**

**Query:**

CREATE OR REPLACE VIEW vw\_updatable\_products AS

SELECT product\_id,product\_name,supplier\_id,category\_id,quantity\_per\_unit,unit\_price,units\_in\_stock,units\_on\_order,reorder\_level,discontinued

FROM

products

WHERE

discontinued = 0;

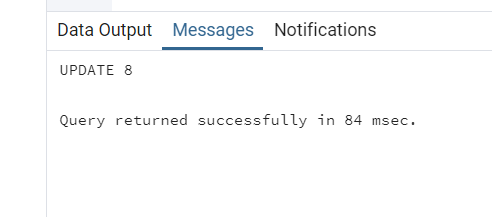
SELECT product\_id, product\_name, unit\_price, units\_in\_stock FROM vw\_updatable\_products WHERE units\_in\_stock < 10;

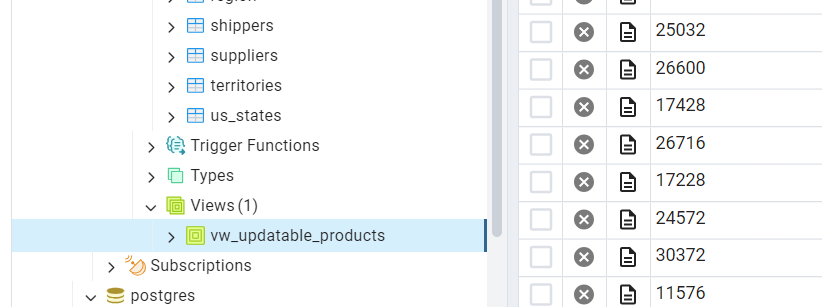
UPDATE vw\_updatable\_products

SET unit\_price = unit\_price \* 1.1

WHERE units\_in\_stock < 10;

**Output:**





**Q2) --2. Transaction:--Update the product price for products by 10% in category id=1.--Try COMMIT and ROLLBACK and observe what happens. -- Start transaction**

**Query:**

BEGIN;

SELECT product\_id, product\_name, unit\_price AS original\_price

FROM products

WHERE category\_id = 1;

UPDATE products

SET limit\_price = unit\_price \* 1.10

WHERE category\_id = 1;

SELECT product\_id, product\_name, unit\_price AS updated\_price

FROM products

WHERE category\_id = 1;

COMMIT;

SELECT product\_id, product\_name, unit\_price AS final\_price

FROM products

WHERE category\_id = 1;

BEGIN;

SELECT product\_id, product\_name, unit\_price AS original\_price

FROM products

WHERE category\_id = 1;

UPDATE products

SET unit\_price = unit\_price \* 1.10

WHERE category\_id = 1;

SELECT product\_id, product\_name, unit\_price AS updated\_price

FROM products

WHERE category\_id = 1;

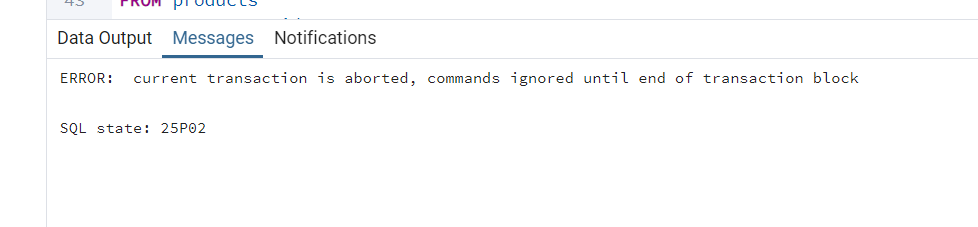
ROLLBACK;

SELECT product\_id, product\_name, unit\_price AS final\_price

FROM products

WHERE category\_id = 1;

**Output:**



**Q3) --3. Create a regular view which will have below details (Need to do joins):--Employee\_id,**

**--Employee\_full\_name,**

**--Title,**

**--Territory\_id,**

**--territory\_description,**

**--region\_description**

**Query:**

CREATE OR REPLACE VIEW vw\_employee\_territories AS

SELECT

e.employee\_id,

CONCAT(e.first\_name, ' ', e.last\_name) AS employee\_full\_name,

e.title,

t.territory\_id,

t.territory\_description,

r.region\_description

FROM

employees e

JOIN

employee\_territories et ON e.employee\_id = et.employee\_id

JOIN

territories t ON et.territory\_id = t.territory\_id

JOIN

region r ON t.region\_id = r.region\_id

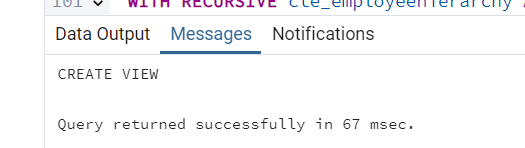
ORDER BY

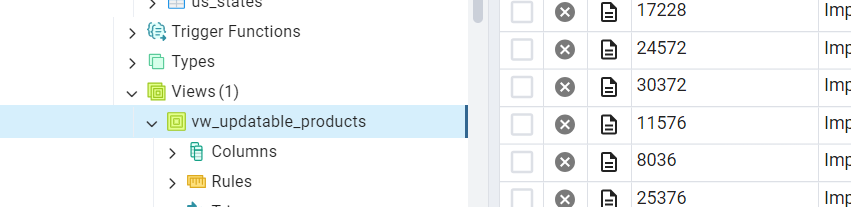
r.region\_description,

t.territory\_description,

employee\_full\_name;

**Output:**





**Q4) --4.Create a recursive CTE based on Employee Hierarchy**

**Query:**

WITH RECURSIVE cte\_employeehierarchy AS (

SELECT

employee\_id,

first\_name,

last\_name,

reports\_to,

0 AS level

FROM

employees

WHERE

reports\_to IS NULL

UNION ALL

SELECT

e.employee\_id,

e.first\_name,

e.last\_name,

e.reports\_to,

eh.level + 1

FROM

employees e

JOIN

cte\_employeehierarchy eh

ON

e.reports\_to = eh.employee\_id

)

SELECT

level,

employee\_id,

first\_name || ' ' || last\_name AS employee\_name

FROM

cte\_employeehierarchy

ORDER BY

level,employee\_id;

SELECT employee\_id, first\_name, reports\_to FROM employees;

Output:

