DAY 8

1.     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;

CREATE VIEW vw\_updatable\_products AS

SELECT product\_id, product\_name, unit\_price, units\_in\_stock

FROM products

WHERE discontinued = 0;

UPDATE vw\_updatable\_products

SET unit\_price = unit\_price \* 1.1

WHERE units\_in\_stock < 10;

SELECT \* FROM products WHERE units\_in\_stock < 10;

CREATE VIEW vw\_updatable\_products AS

SELECT product\_id, product\_name, unit\_price, units\_in\_stock

FROM products

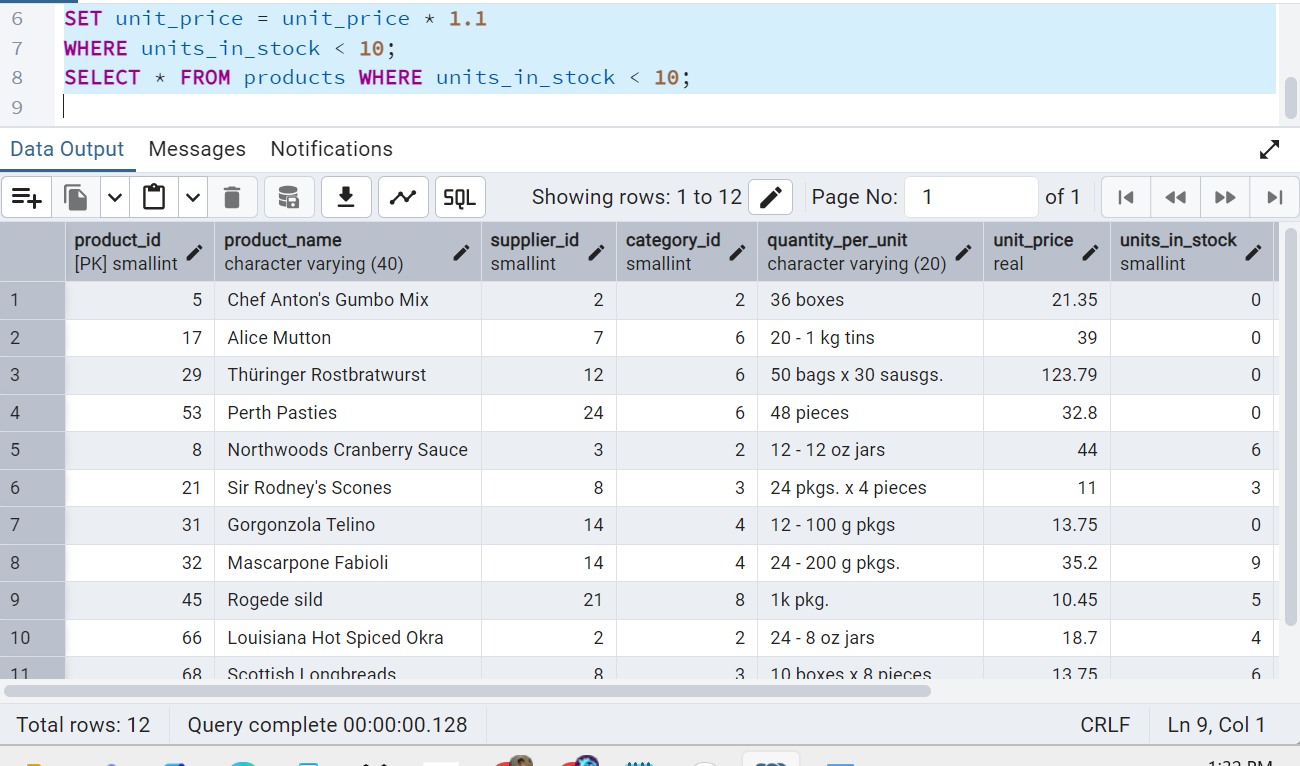
WHERE discontinued = 0;

UPDATE vw\_updatable\_products

SET unit\_price = unit\_price \* 1.1

WHERE units\_in\_stock < 10;

SELECT \* FROM products WHERE units\_in\_stock < 10;



2.     Transaction:

Update the product price for products by 10% in category id=1

Try COMMIT and ROLLBACK and observe what happens.

BEGIN; -- Start a transaction

UPDATE products

SET unit\_price = unit\_price \* 1.1

WHERE category\_id = 1;

-- Check changes BEFORE commit (optional)

SELECT \* FROM products WHERE category\_id = 1;

COMMIT; -- Finalize the transaction (changes are saved)

BEGIN; -- Start a transaction

UPDATE products

SET unit\_price = unit\_price \* 1.1

WHERE category\_id = 1;

-- Check changes BEFORE rollback (optional)

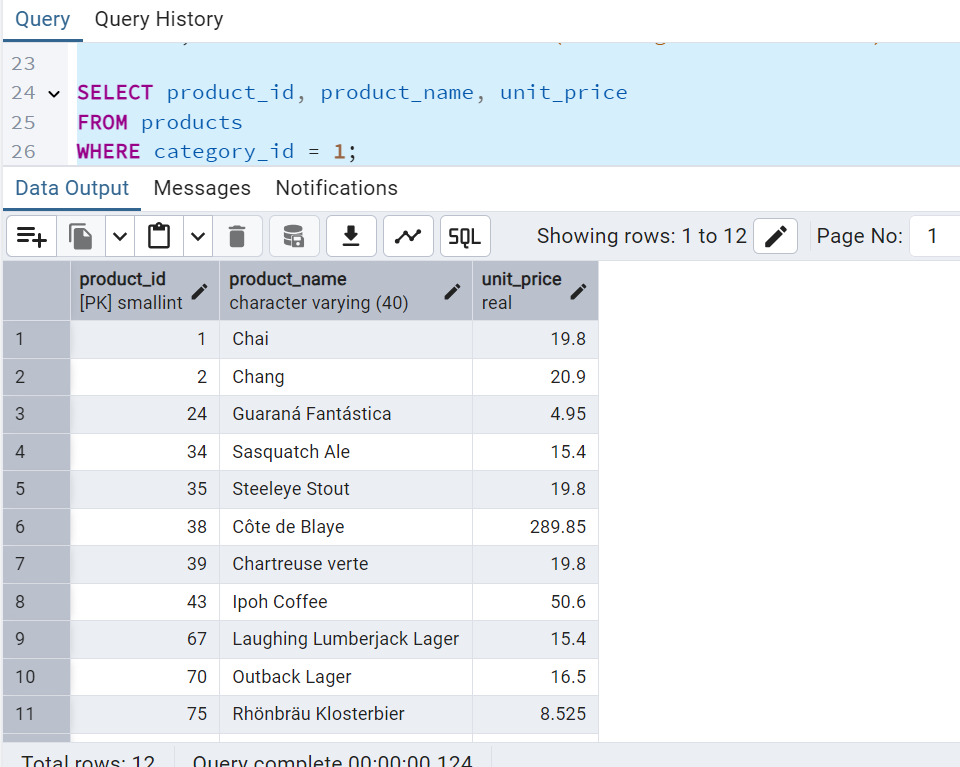
SELECT \* FROM products WHERE category\_id = 1;

ROLLBACK; -- Cancel the transaction (no changes will be saved)

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

FROM products

WHERE category\_id = 1;



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

CREATE VIEW vw\_employee\_territories AS

SELECT

e.employee\_id,

e.first\_name || ' ' || e.last\_name AS employee\_full\_name, -- For SQL Server use: e.first\_name + ' ' + e.last\_name

e.title,

t.territory\_id,

t.territory\_description,

r.region\_description

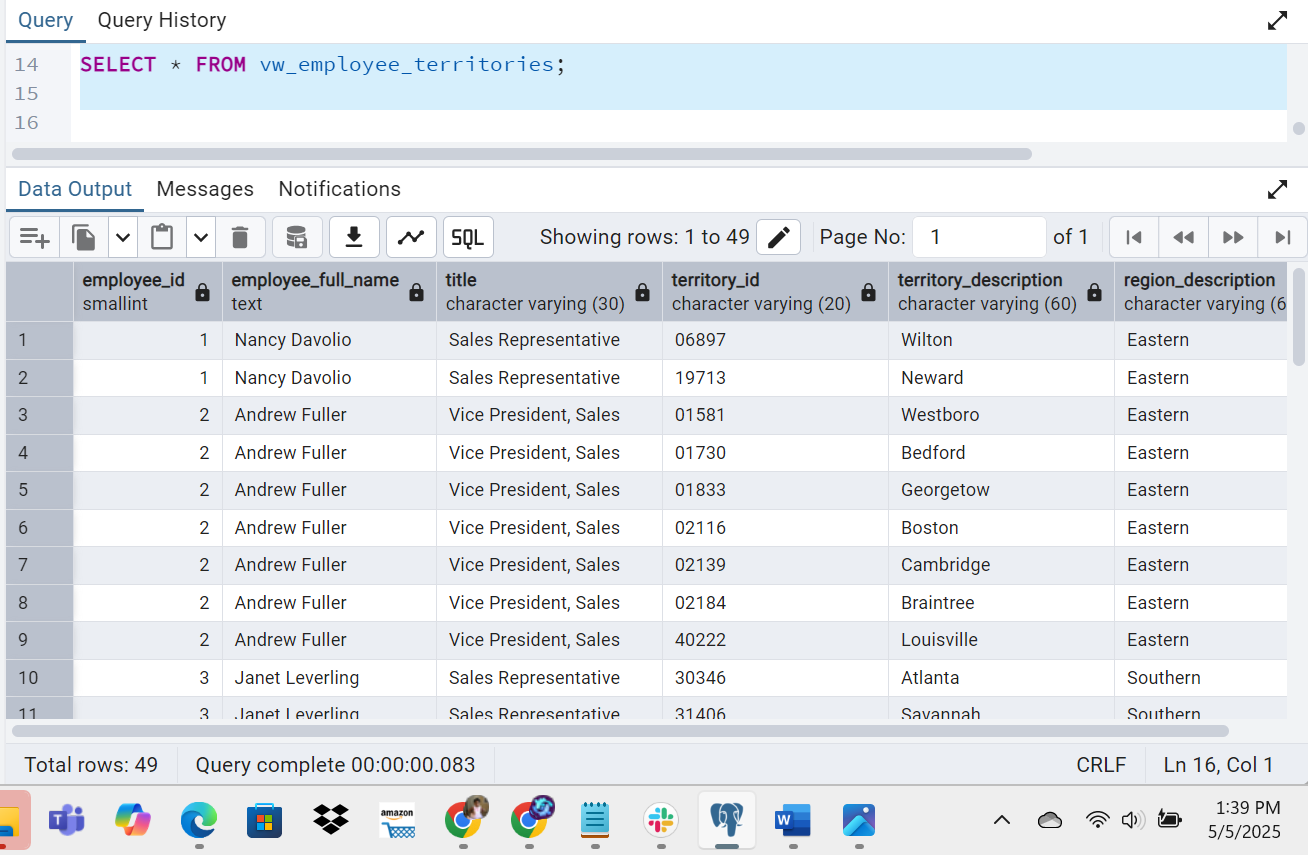
FROM employees e

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;

SELECT \* FROM vw\_employee\_territories;



4.     Create a recursive CTE based on Employee Hierarchy

WITH RECURSIVE employee\_hierarchy AS (

-- Anchor member: top-level managers (no one they report to)

SELECT

employee\_id,

first\_name,

last\_name,

reports\_to,

1 AS level

FROM employees

WHERE reports\_to IS NULL

UNION ALL

-- Recursive member: employees who report to others

SELECT

e.employee\_id,

e.first\_name,

e.last\_name,

e.reports\_to,

eh.level + 1

FROM employees e

INNER JOIN employee\_hierarchy eh ON e.reports\_to = eh.employee\_id

)

SELECT \* FROM employee\_hierarchy

ORDER BY level, employee\_id;

