**SQL Final Project Submission :**

**Name** : Snehal Yerkuntwar (snehalyr)

**LINK to onecompiler** : <https://onecompiler.com/postgresql/43tysjf3p>

**NOTE** : In myCompiler , The Data Creation and Data Entry Part are common for all sub-tasks. The SQL codes and their explanation for all the (Questions and its Answer) tasks are in the same file. You need to scroll down to arrive at every question and its SQL query along with its explanation. The main query or code is Inactivated by “- -”.

So whenever you have to check the query, just remove “- -” before ‘SELECT /INSERT’ command and verify it on Output prompt, and then reapply “- -” so thaonet it does not overlap on other queries.

You will find comment “ Task Begins from Here” and from there apply as mentioned above.

**Title: Task 3: Online Store Order Management System (PostgreSQL)**

**Objective:** Create a system to manage orders, customers, and products for an online store.

The project will include the following tasks:

**Database Creation:**

Create a database named OnlineStore.

Create tables:

Customers (CUSTOMER\_ID, NAME, EMAIL, PHONE, ADDRESS)

Products (PRODUCT\_ID, PRODUCT\_NAME, CATEGORY, PRICE, STOCK)

Orders (ORDER\_ID, CUSTOMER\_ID, PRODUCT\_ID, QUANTITY, ORDER\_DATE) Set up foreign keys linking Orders to Customers and Products.

**Data Creation**: Insert sample data for customers, products, and orders

1. **Order Management:**

a) Retrieve all orders placed by a specific customer.

b) Find products that are out of stock.

c) Calculate the total revenue generated per product.

d) Retrieve the top 5 customers by total purchase amount.

e) Find customers who placed orders in at least two different product categories.

**B. Analytics:**

a) Find the month with the highest total sales.

b) Identify products with no orders in the last 6 months.

c) Retrieve customers who have never placed an order.

d) Calculate the average order value across all orders.

**Main Code: Task wise begins from here**

**Database Creation:**

---- CREATE DATABASE OnlineStore;: This creates the database named "OnlineStore".

---- USE OnlineStore;: This switches the active database to "OnlineStore"

CREATE DATABASE OnlineStore;

USE OnlineStore;

**Table Creation:**

* CREATE TABLE Customers (...): Creates the Customers table with columns for customer ID, name, email, phone, and address. CUSTOMER\_ID is set as the primary key.
* CREATE TABLE Products (...): Creates the Products table with columns for product ID, name, category, price, and stock. PRODUCT\_ID is set as the primary key.
* CREATE TABLE Orders (...): Creates the Orders table with columns for order ID, customer ID, product ID, quantity, and order date. ORDER\_ID is set as the primary key. Foreign keys are established to link CUSTOMER\_ID to the Customers table and PRODUCT\_ID to the Products table, ensuring referential integrity.

CREATE TABLE Customers (

CUSTOMER\_ID INT PRIMARY KEY,

NAME VARCHAR(255),

EMAIL VARCHAR(255),

PHONE VARCHAR(20),

ADDRESS VARCHAR(255)

);

CREATE TABLE Products (

PRODUCT\_ID INT PRIMARY KEY,

PRODUCT\_NAME VARCHAR(255),

CATEGORY VARCHAR(255),

PRICE DECIMAL(10, 2),

STOCK INT

);

CREATE TABLE Orders (

ORDER\_ID INT PRIMARY KEY,

CUSTOMER\_ID INT,

PRODUCT\_ID INT,

QUANTITY INT,

ORDER\_DATE DATE,

FOREIGN KEY (CUSTOMER\_ID) REFERENCES Customers(CUSTOMER\_ID),

FOREIGN KEY (PRODUCT\_ID) REFERENCES Products(PRODUCT\_ID)

);

**Data Creation (Sample Data):**

INSERT INTO Customers (CUSTOMER\_ID, NAME, EMAIL, PHONE, ADDRESS) VALUES

(1, 'John Doe', 'john.doe@email.com', '555-123-4567', '123 Main St'),

(2, 'Jane Smith', 'jane.smith@email.com', '555-987-6543', '456 Oak Ave'),

(3, 'Peter Jones', 'peter.jones@email.com', '555-246-8013', '789 Pine Ln');

INSERT INTO Products (PRODUCT\_ID, PRODUCT\_NAME, CATEGORY, PRICE, STOCK) VALUES

(101, 'Laptop', 'Electronics', 1200.00, 50),

(102, 'Tablet', 'Electronics', 300.00, 75),

(201, 'T-shirt', 'Apparel', 25.00, 200),

(202, 'Jeans', 'Apparel', 75.00, 150);

INSERT INTO Orders (ORDER\_ID, CUSTOMER\_ID, PRODUCT\_ID, QUANTITY, ORDER\_DATE) VALUES

(1, 1, 101, 1, '2025-08-01'),

(2, 1, 202, 2, '2025-08-05'),

(3, 2, 201, 3, '2025-08-10'),

(4, 3, 102, 1, '2025-08-15');

* INSERT INTO Customers (...) VALUES (...): Inserts sample data for three customers.
* INSERT INTO Products (...) VALUES (...): Inserts sample data for four products, two in "Electronics" and two in "Apparel".
* INSERT INTO Orders (...) VALUES (...): Inserts sample data for four orders, linking customers to products and including order dates.

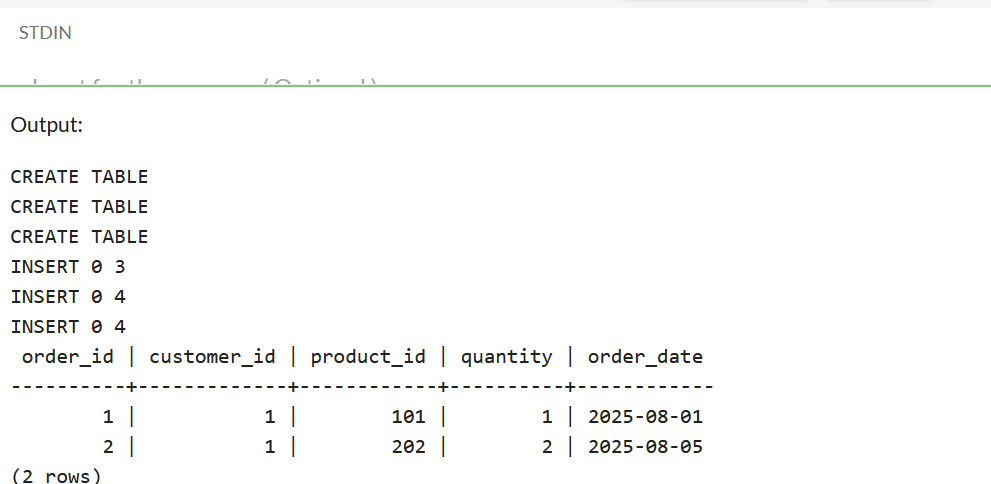
**A. Order Management**

a**) Retrieve all orders placed by a specific customer.**

SELECT \* FROM Orders WHERE CUSTOMER\_ID = 1;

*---- Replace 1 with the desired CUSTOMER\_ID*

---- This query retrieves all columns (\*) from the Orders table where the CUSTOMER\_ID matches the specified ID (e.g., 1).



**b) Find products that are out of stock**.

-- DATA Manipulation Required

-- Adding Zero Stock products to our sample data as Data Manipulation Step

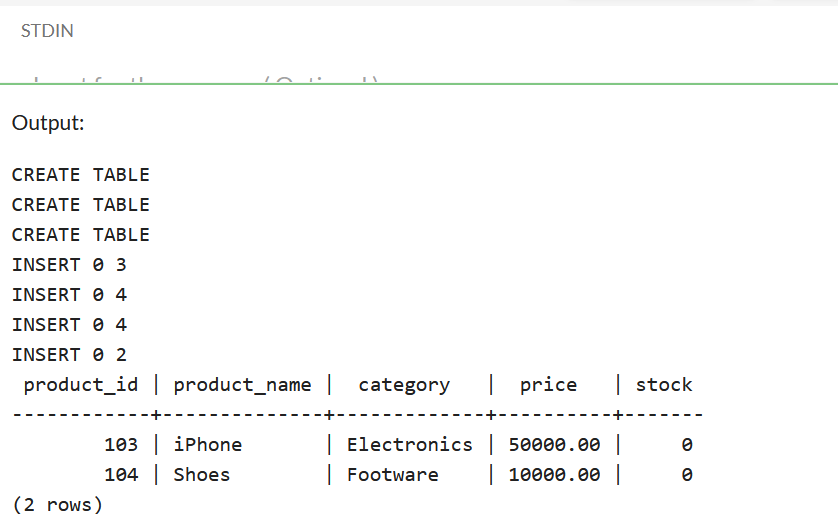
INSERT INTO Products (PRODUCT\_ID, PRODUCT\_NAME, CATEGORY, PRICE, STOCK) VALUES

(103, 'iPhone', 'Electronics', 50000.00, 0),

(104, 'Shoes', 'Footware', 10000.00, 0);

SELECT \* FROM Products WHERE STOCK = 0;

---- This query retrieves all columns (\*) from the Products table where the STOCK is equal to 0.



**c) Calculate the total revenue generated per product.**

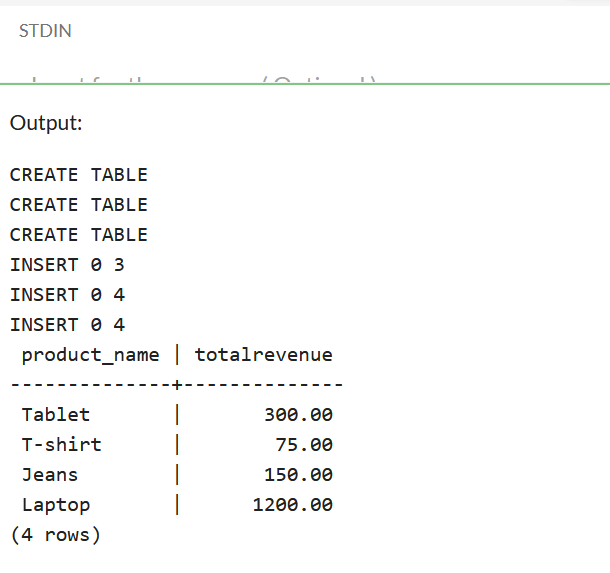
SELECT p.PRODUCT\_NAME, SUM(o.QUANTITY \* p.PRICE) AS TotalRevenue

FROM Products p

JOIN Orders o ON p.PRODUCT\_ID = o.PRODUCT\_ID

GROUP BY p.PRODUCT\_NAME;

---- This query joins the Products and Orders tables on PRODUCT\_ID. It then calculates the TotalRevenue for each product by multiplying the QUANTITY from Orders with the PRICE from Products, and groups the results by PRODUCT\_NAME to provide the revenue per product.



**d) Retrieve the top 5 customers by total purchase amount.**

--- Data Manipulation required for Orders and Customers Table as the sample data is insufficient to get required Output as per the query

INSERT INTO Customers (CUSTOMER\_ID, NAME, EMAIL, PHONE, ADDRESS) VALUES

(4, 'Alice Smith', 'alice.s@example.com', '123-456-7890','123 Main St'),

(5, 'Bob Johnson', 'bob.j@example.com', '098-765-4321' ,'123 Main St'),

(6, 'Frank White', 'frank.w@example.com', '222-333-4444','123 Main St');

INSERT INTO Orders (ORDER\_ID, CUSTOMER\_ID, PRODUCT\_ID, QUANTITY, ORDER\_DATE) VALUES

(5, 3, 102, 1, '2025-08-15'),

(6, 2, 102, 1, '2025-08-15'),

(7, 3, 101, 2, '2025-08-15'),

(8, 4, 102, 4, '2025-08-15'),

(9, 5, 202, 2, '2025-08-15'),

(10, 6, 201, 5, '2025-08-15');

SELECT c.NAME, SUM(o.QUANTITY \* p.PRICE) AS TotalSpent

FROM Customers c

JOIN Orders o ON c.CUSTOMER\_ID = o.CUSTOMER\_ID

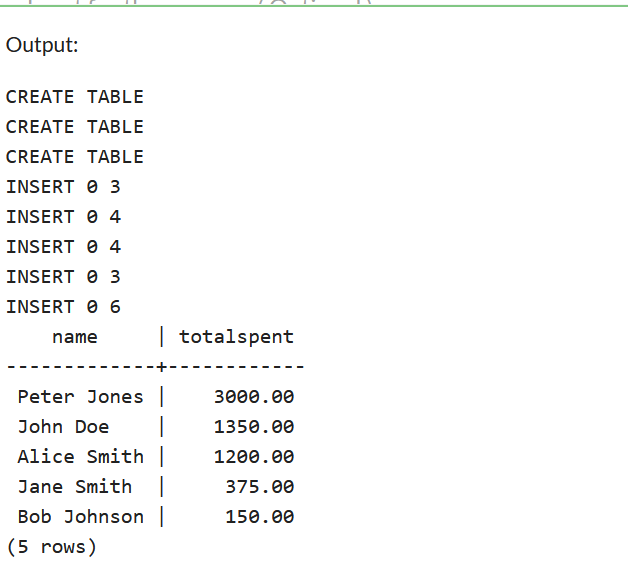
JOIN Products p ON o.PRODUCT\_ID = p.PRODUCT\_ID

GROUP BY c.NAME

ORDER BY TotalSpent DESC

LIMIT 5;

---- This query joins Customers, Orders, and Products tables to calculate the TotalSpent for each customer. It groups the results by customer NAME, orders them in descending order of TotalSpent, and limits the output to the top 5 customers



e) Find customers who placed orders in at least two different product categories.

--- Adding More sample data to Customer, Product and Orders tables specifically to get more data as sample data is insufficient to get output for this query

INSERT INTO Products (PRODUCT\_ID, PRODUCT\_NAME, CATEGORY, PRICE, STOCK) VALUES

(103, 'iPhone', 'Electronics', 50000.00, 0),

(104, 'Shoes', 'Footware', 10000.00, 0);

INSERT INTO Customers (CUSTOMER\_ID, NAME, EMAIL, PHONE, ADDRESS) VALUES

(4, 'Alice Smith','alice.s@example.com', '123-456-7890','123 Main St'),

(5, 'Bob Johnson', 'bob.j@example.com', '098-765-4321' ,'123 Main St'),

(6, 'Frank White', 'frank.w@example.com', '222-333-4444','123 Main St');

INSERT INTO Orders (ORDER\_ID, CUSTOMER\_ID, PRODUCT\_ID, QUANTITY, ORDER\_DATE) VALUES

(5, 3, 102, 1, '2025-08-15'),

(6, 3, 202, 1, '2025-08-15'),

(7, 4, 201, 2, '2025-08-15'),

(8, 4, 102, 4, '2025-08-15'),

(9, 5, 202, 2, '2025-08-15'),

(10, 5, 201, 5, '2025-08-15');

SELECT c.NAME

FROM Customers c

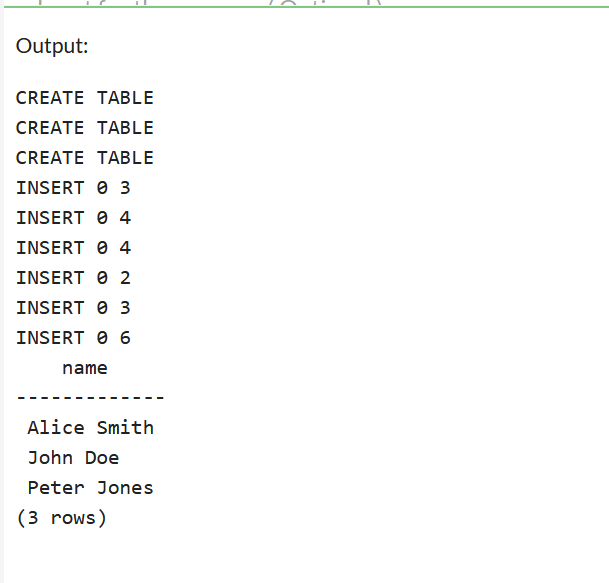
JOIN Orders o ON c.CUSTOMER\_ID = o.CUSTOMER\_ID

JOIN Products p ON o.PRODUCT\_ID = p.PRODUCT\_ID

GROUP BY c.NAME

HAVING COUNT(DISTINCT p.CATEGORY) >= 2;

---- This query joins Customers, Orders, and Products tables. It groups the results by customer NAME and uses the HAVING clause to filter out customers who have ordered from at least two distinct product CATEGORY values.



**B. Analytics:**

a) Find the month with the highest total sales

SELECT

EXTRACT(YEAR FROM o.ORDER\_DATE) AS SaleYear,

EXTRACT(MONTH FROM o.ORDER\_DATE) AS SaleMonth,

SUM(o.QUANTITY \* p.PRICE) AS TotalSales

FROM

Orders o

JOIN

Products p ON o.PRODUCT\_ID = p.PRODUCT\_ID

GROUP BY

SaleYear,

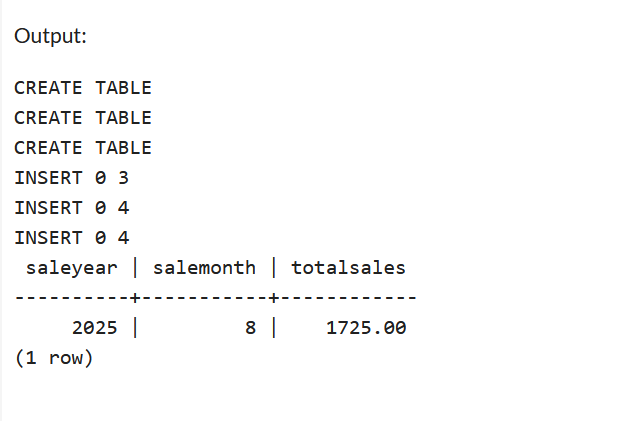
SaleMonth

ORDER BY

TotalSales DESC

LIMIT 1;

* EXTRACT(YEAR FROM o.ORDER\_DATE) AS SaleYear: Extracts the year from the ORDER\_DATE column and labels it SaleYear.
* EXTRACT(MONTH FROM o.ORDER\_DATE) AS SaleMonth: Extracts the month from the ORDER\_DATE column and labels it SaleMonth.
* SUM(o.QUANTITY \* p.PRICE) AS TotalSales: Calculates the total sales for each order by multiplying the QUANTITY with the PRICE, then sums these values for each month.
* FROM Orders o JOIN Products p ON o.PRODUCT\_ID = p.PRODUCT\_ID: Joins the Orders and Products tables based on the common PRODUCT\_ID column.
* GROUP BY SaleYear, SaleMonth: Groups the results by year and month to aggregate sales data for each month.



**b) Identify products with no orders in the last 6 months.**

UPDATE Orders

SET ORDER\_DATE = '2024-01-15' *-- An order date more than 6 months ago*

WHERE ORDER\_ID = 1;

UPDATE Orders

SET ORDER\_DATE = '2024-02-20' *-- Another order date more than 6 months ago*

WHERE ORDER\_ID = 2;

*-- Keep some orders within the last 6 months for comparison*

*-- (The other orders are already recent enough)*

SELECT

p.PRODUCT\_ID,

p.PRODUCT\_NAME

FROM

Products p

LEFT JOIN

Orders o ON p.PRODUCT\_ID = o.PRODUCT\_ID

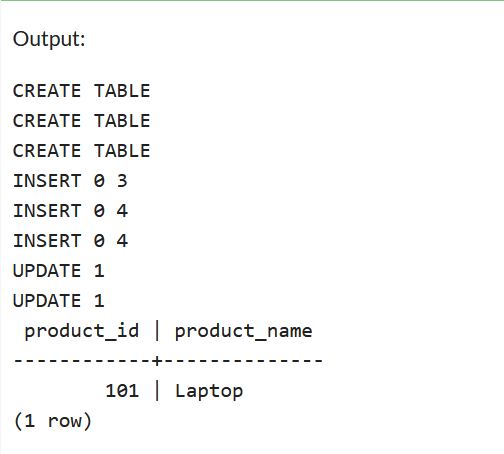
AND o.ORDER\_DATE >= NOW() - INTERVAL '6 months'

WHERE

o.ORDER\_ID IS NULL;

Explanation

* WHERE NOT EXISTS (...):
  + This is the main filtering condition.
* SELECT 1 FROM Orders o WHERE o.PRODUCT\_ID = p.PRODUCT\_ID AND o.ORDER\_DATE >= NOW() - INTERVAL '6 months':
  + This subquery attempts to find any order for the current product (p.PRODUCT\_ID) that occurred within the last 6 months.
  + If the subquery finds *any* such order (it doesn't matter what it selects, so SELECT 1 is common and efficient), EXISTS would return true.
  + NOT EXISTS then ensures that the outer query only returns products for which this subquery finds no matching orders.



c) Retrieve customers who have never placed an order.

-- Inserting sample data for Customers table for Zero order placed by customer as

previously inserted data is insufficient

INSERT INTO Customers (CUSTOMER\_ID, NAME, EMAIL, PHONE, ADDRESS) VALUES

(4, 'Alice Smith','alice.s@example.com', '123-456-7890','123 Main St'),

(5, 'Bob Johnson', 'bob.j@example.com', '098-765-4321' ,'123 Main St'),

(6, 'Frank White', 'frank.w@example.com', '222-333-4444','123 Main St');

SELECT c.NAME

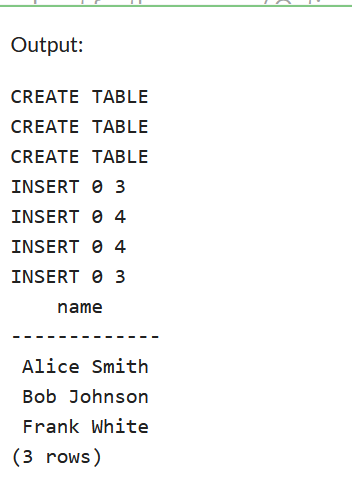
FROM Customers c

LEFT JOIN Orders o ON c.CUSTOMER\_ID = o.CUSTOMER\_ID

WHERE o.CUSTOMER\_ID IS NULL;

-- This query uses a LEFT JOIN to combine the Customers table with the Orders table on CUSTOMER\_ID.

-- The WHERE clause filters for customers where their CUSTOMER\_ID in the Orders table is NULL, indicating they have no corresponding orders



**d) Calculate the average order value across all orders.**

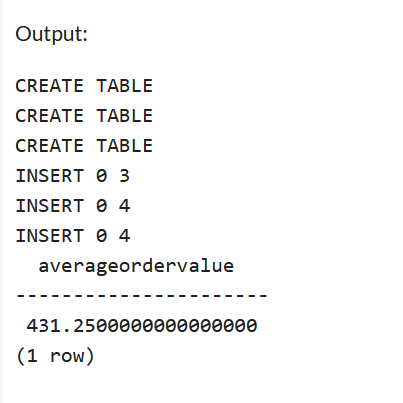
SELECT AVG(o.QUANTITY \* p.PRICE) AS AverageOrderValue

FROM Orders o

JOIN Products p ON o.PRODUCT\_ID = p.PRODUCT\_ID;

-- This query first joins the Orders and Products tables to access both the QUANTITY from orders and the PRICE of the ordered products.

-- The AVG function then calculates the average of the product of QUANTITY and PRICE for all orders, representing the average order value.



**END OF TASK 3**