**TASK 3.**

**-- Sample Data for Customers Table**

create table customers(

customer\_id serial primary key,

first\_name varchar(50),

last\_name varchar(50),

city varchar(100),

country varchar(100)

);

INSERT INTO Customers (customer\_id, first\_name, last\_name, city, country) VALUES

(1, 'John', 'Doe', 'London', 'UK'),

(2, 'Jane', 'Smith', 'New York', 'USA'),

(3, 'Peter', 'Jones', 'London', 'UK'),

(4, 'Alice', 'Brown', 'Paris', 'France'),

(5, 'Bob', 'Williams', 'New York', 'USA');

**select \* from cutomers**

-- Sample Data for Orders Table

create table orders(

order\_id serial primary key,

customer\_id varchar(50),

order\_date date,

total\_amount numeric(100,2)

);

INSERT INTO Orders (order\_id, customer\_id, order\_date, total\_amount) VALUES

(101, 1, '2025-04-01', 45.50),

(102, 2, '2025-04-03', 120.00),

(103, 1, '2025-04-05', 78.20),

(104, 3, '2025-04-07', 30.00),

(105, 2, '2025-04-09', 95.75);

select \* from orders

**-- Sample Data for Books Table**

create table Books(

book\_id serial primary key,

title varchar(50),

author varchar(100),

price numeric(100,2)

);

INSERT INTO Books (book\_id, title, author, price) VALUES

(1, 'The Great Novel', 'Author A', 19.99),

(2, 'Mystery in the Woods', 'Author B', 12.50),

(3, 'Coding for Beginners', 'Author C', 25.00),

(4, 'Historical Journey', 'Author D', 15.75);

select \* from Books

**-- Sample Data for OrderItems Table**

create table OrderItems(

order\_item\_id serial primary key,

order\_id int,

book\_id int,

quantity int,

item\_price numeric(100,2)

);

INSERT INTO OrderItems (order\_item\_id, order\_id, book\_id, quantity, item\_price) VALUES

(1, 101, 1, 1, 19.99),

(2, 101, 2, 1, 12.50),

(3, 102, 3, 2, 25.00),

(4, 102, 4, 1, 15.75),

(5, 103, 1, 3, 19.99),

(6, 104, 2, 2, 12.50),

(7, 105, 3, 1, 25.00),

(8, 105, 1, 1, 19.99);

select \* from OrderItems

**-- a. Use SELECT, WHERE, ORDER BY, GROUP BY**

**-- 1. Select all customers from the city 'London', ordered by last name.**

SELECT \*

FROM Customers

WHERE city = 'London'

ORDER BY last\_name;

**-- 2. Find the total number of orders placed each day.**

SELECT order\_date, COUNT(\*) AS total\_orders

FROM Orders

GROUP BY order\_date

ORDER BY order\_date;

-- b. Use JOINS (INNER, LEFT, RIGHT)

**-- 3. Get the first name of customers and the order IDs they have placed (INNER JOIN).**

SELECT c.first\_name, o.order\_id

FROM Customers c

INNER JOIN Orders o ON c.customer\_id = o.customer\_id;

**-- 4. Get all customers and their order IDs. If a customer has not placed any orders, their order ID will be NULL (LEFT JOIN).**

SELECT c.first\_name, o.order\_id

FROM Customers c

LEFT JOIN Orders o ON c.customer\_id = o.customer\_id;

**-- 5. Get all orders and the first name of the customer who placed them. If an order has no associated customer (which shouldn't happen with proper FK constraints, but for demonstration), the customer's first name will be NULL (RIGHT JOIN).**

SELECT o.order\_id, c.first\_name

FROM Orders o

RIGHT JOIN Customers c ON o.customer\_id = c.customer\_id;

**-- c. Write subqueries**

**-- 6. Find the customers who have placed at least one order with a total amount greater than 50.**

SELECT first\_name, last\_name

FROM Customers

WHERE customer\_id IN (SELECT customer\_id FROM Orders WHERE total\_amount > 50);

**-- 7. Find the books that have a price higher than the average price of all books.**

SELECT title, price

FROM Books

WHERE price > (SELECT AVG(price) FROM Books);

**-- d. Use aggregate functions (SUM, AVG)**

**-- 8. Calculate the total revenue from all orders.**

SELECT SUM(total\_amount) AS total\_revenue

FROM Orders;

**-- 9. Calculate the average price of books.**

SELECT AVG(price) AS average\_book\_price

FROM Books;

**-- 10. Find the customer who has spent the most money.**

SELECT c.first\_name, c.last\_name, SUM(o.total\_amount) AS total\_spent

FROM Customers c

JOIN Orders o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id, c.first\_name, c.last\_name

ORDER BY total\_spent DESC

LIMIT 1;

**-- e. Create views for analysis**

**-- 11. Create a view showing customer names and their total order amounts.**

CREATE VIEW CustomerOrderTotals AS

SELECT c.first\_name, c.last\_name, SUM(o.total\_amount) AS total\_spent

FROM Customers c

JOIN Orders o ON c.customer\_id = o.customer\_id

GROUP BY c.customer\_id, c.first\_name, c.last\_name;

**-- Now you can query the view:**

SELECT \* FROM CustomerOrderTotals ORDER BY total\_spent DESC;

**-- 12. Create a view showing book titles and the total quantity ordered.**

CREATE VIEW BookOrderQuantities AS

SELECT b.title, SUM(oi.quantity) AS total\_quantity\_ordered

FROM Books b

JOIN OrderItems oi ON b.book\_id = oi.book\_id

GROUP BY b.book\_id, b.title;

**-- Now you can query the view:**

SELECT \* FROM BookOrderQuantities ORDER BY total\_quantity\_ordered DESC;

**-- f. Optimize queries with indexes**

-- Assuming you have a large dataset, adding indexes to frequently used columns in WHERE clauses and JOIN conditions can improve query performance.

**-- 13. Add an index on the 'city' column of the Customers table.**

CREATE INDEX idx\_city ON Customers (city);

**-- 14. Add an index on the 'customer\_id' column of the Orders table (as it's a foreign key and used in joins).**

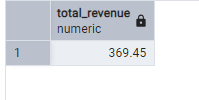
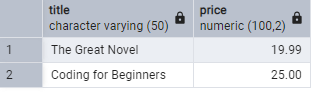
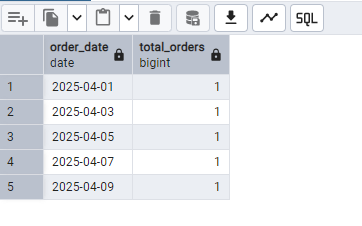
CREATE INDEX idx\_customer\_id ON Orders (customer\_id);

**-- 15. Add a composite index on 'order\_id' and 'book\_id' in the OrderItems table.**

CREATE INDEX idx\_order\_book ON OrderItems (order\_id, book\_id);

**Here is SS of output:**

**A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.**