Create the following tables, insert data, and generate the queries which are given below:

Tables with Data

Table: Customers

CustomerID	Name	City	JoinDate	CreditLimit
1	Alice Johnson	New York	2023-01-15	10000
2	Bob Smith	Los Angeles	2022-10-10	15000
3	Charlie Brown	Chicago	2024-05-05	8000
4	David Wilson	New York	2023-08-20	20000
5	Emma Thomas	Miami	2023-02-12	12000

Table: Orders

OrderID	CustomerID	OrderDate	TotalAmount
101	1	2024-01-01	1200
102	3	2024-06-15	2500
103	2	2024-03-20	1800
104	4	2024-02-10	3000
105	1	2024-04-05	2200
106	5	2024-05-18	3500

Table: Products

ProductID	ProductName	Category	Price
201	Laptop	Electronics	1200
202	Smartphone	Electronics	800
203	Office Chair	Furniture	150
204	Dining Table	Furniture	600
205	Headphones	Electronics	300

Table: OrderDetails

OrderDetailID	OrderID	ProductID	Quantity	UnitPrice
1	101	201	1	50000
2	101	202	2	25000
3	102	203	4	5000
4	103	204	1	15000
5	104	205	3	3000
6	105	201	2	50000
7	106	202	1	25000

1. Simple Queries

1. Retrieve all columns from the **Customers** table.
2. List all products in the **'Furniture'** category.
3. Display orders where the total amount is greater than **₹3000**.
4. Find all customers who joined before **2023-06-01**.
5. Retrieve the names and cities of customers who have placed at least one order.

---

2. `IN`, `LIKE`, and Date Functions

6. List customers from cities **'Mumbai'**, **'Delhi'**, or **'Chennai'** using the `IN` clause.
7. Find product names that contain the word **'Table'** using the `LIKE` keyword.
8. Retrieve orders placed in the month of **April 2024** using a date function.
9. List products whose price is between **₹5000** and **₹30000** using `BETWEEN`.
10. Find customers whose names start with the letter **'R'**.

---

3. Aggregate Functions

11. Count the total number of orders placed by each customer using `GROUP BY`.
12. Find the average price of products in each category.
13. Retrieve the maximum and minimum order total amounts.
14. Compute the total revenue generated from all orders.
15. Find the total number of distinct products ordered.

---

4. `GROUP BY`, `HAVING`, and `DISTINCT`

16. List the total number of orders placed by customers who have placed more than **1 order** using `HAVING`.
17. Retrieve the distinct categories of products available.
18. Display the total revenue generated by each customer, only including those who have generated more than **₹4000** in revenue.
19. Count the number of products sold in each category.
20. List distinct cities where customers are located.

---

5. `ORDER BY` (Multi-column)

21. Retrieve products ordered by **Category** in ascending order and **Price** in descending order.
22. Display customers sorted by **City** in ascending order and **CreditLimit** in descending order.
23. List orders sorted by **OrderDate** and **TotalAmount**.
24. Find the top 3 most expensive products using `ORDER BY` and `LIMIT`.
25. Retrieve the most recent order placed by each customer using `GROUP BY` and `ORDER BY`.

---

6. Set Operations

26. Retrieve the names of customers who have placed orders and those who haven't using a `UNION` of two queries.
27. Find customers who have placed orders for products in the **'Furniture'** category using the `INTERSECT` set operation.
28. List customers who have placed orders, but exclude those whose total order amount is less than **₹2000** using the `EXCEPT` set operation.

---

7. Joins and Subqueries

29. Retrieve customer names along with their order IDs and product names using an **inner join** of **Customers**, **Orders**, and **OrderDetails**.
30. Find the names of customers who have placed orders with a total amount greater than the average order amount using a **subquery**.
31. Display the total quantity of products ordered by each customer using a **join** between **Orders** and **OrderDetails**, and group by customer name.
32. List customers who have never placed an order using a **left join** and filtering for null order IDs.
33. Retrieve product names along with the total revenue generated for each product using a **join** and aggregate functions.
34. Find pairs of customers from the same city using a **self-join** on the **Customers** table.