**Project Overview**

This project focuses on developing a structured SQL database for managing customers, orders, products, and payments. It involves writing and executing SQL queries to extract insights, manage data efficiently, and understand the relationships between different tables.

**Objectives**

* Implement database design principles.
* Retrieve and manipulate data using SQL queries.
* Utilize aggregate functions for analytical insights.
* Apply join operations to combine data from multiple tables.
* Use subqueries and advanced filters for efficient data processing.

**Database Schema**

The project involves the following tables:

1. **Customers**: Stores customer details.
2. **Orders**: Contains order information.
3. **Products**: Lists products available in the store.
4. **Order\_Items**: Stores items included in each order.
5. **Payments**: Maintains records of payments made for orders.

**Practice Questions & SQL Queries**

**1️. Basic Data Retrieval (SELECT, WHERE, ORDER BY, LIMIT)**

1. Retrieve all records from the **customers** table.

SELECT \* FROM customers;

1. Fetch only the **customer ID, first name, and email** from the **customers** table.

SELECT customer\_id, first\_name, email FROM customers;

1. List all **products** in the **Clothing** category.

SELECT \* FROM products WHERE category = 'Clothing';

1. Retrieve all **orders** where the total amount is **greater than $500**.

SELECT \* FROM orders WHERE total\_amount > 500;

1. Find all **customers who joined after January 1, 2023**.

SELECT \* FROM customers WHERE join\_date > '2023-01-01';

1. Display the **top 5 most expensive products**.

SELECT \* FROM products ORDER BY price DESC LIMIT 5;

1. List the **latest 10 orders** placed, sorted by order date in **descending order**.

SELECT \* FROM orders ORDER BY order\_date DESC LIMIT 10;

1. Retrieve all **orders** that have a status of **"Completed"**.

SELECT \* FROM orders WHERE status = 'Completed';

1. Find all **orders** that were placed between **February 1, 2023, and February 28, 2023**.

SELECT \* FROM orders WHERE order\_date BETWEEN '2023-02-01' AND '2023-02-28';

1. List all **products** that have a **price between $50 and $100**.

SELECT \* FROM products WHERE price BETWEEN 50 AND 100;

**2️. Aggregate Functions (COUNT, SUM, AVG, MIN, MAX, GROUP BY, HAVING)**

1. Count the **total number of customers** in the database.

SELECT COUNT(\*) FROM customers;

1. Find the **average order amount**.

SELECT AVG(total\_amount) FROM orders;

1. Retrieve the **highest and lowest** priced products.

SELECT MAX(price), MIN(price) FROM products;

1. Count the **number of products per category**.

SELECT category, COUNT(\*) FROM products GROUP BY category;

1. Calculate the **total revenue from all orders**.

SELECT SUM(total\_amount) FROM orders;

1. Find the **total number of orders placed by each customer**, sorted by highest to lowest.

SELECT customer\_id, COUNT(\*) FROM orders GROUP BY customer\_id ORDER BY COUNT(\*) DESC;

1. Calculate the **total revenue generated for each month** in 2023.

SELECT MONTH(order\_date) AS month, SUM(total\_amount) FROM orders WHERE YEAR(order\_date) = 2023 GROUP BY MONTH(order\_date);

1. List all **customers who have placed more than 5 orders**.

SELECT customer\_id, COUNT(\*) FROM orders GROUP BY customer\_id HAVING COUNT(\*) > 5;

1. Identify the **most frequently used payment method** based on the number of transactions.

SELECT payment\_method, COUNT(\*) FROM payments GROUP BY payment\_method ORDER BY COUNT(\*) DESC LIMIT 1;

1. Find the **average product price** for each category.

SELECT category, AVG(price) FROM products GROUP BY category;

**3️⃣ Joins (INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN)**

1️⃣ Retrieve all **order details along with the customer’s first and last name**.  
2️⃣ Fetch **order items with product names, quantities, and subtotal values**.  
3️⃣ List all **payment transactions along with the corresponding order details**.  
4️⃣ Identify **customers who have never placed an order**.  
5️⃣ Find all **products that have never been purchased** (i.e., do not appear in any order).  
6️⃣ Retrieve **customers and their total spending** by summing up all their orders.  
7️⃣ Get the **total number of products ordered by each customer**.  
8️⃣ Display all **orders along with the names of the products included in each order**.  
9️⃣ Find **orders that do not have any associated payments** recorded.  
🔟 Retrieve **customers along with the last date they placed an order**.

SELECT o.order\_id, c.first\_name, c.last\_name, o.total\_amount FROM orders o JOIN customers c ON o.customer\_id = c.customer\_id;

SELECT oi.order\_id, p.product\_name, oi.quantity, oi.subtotal FROM order\_items oi JOIN products p ON oi.product\_id = p.product\_id;

SELECT p.payment\_id, o.order\_id, p.payment\_method, p.payment\_status FROM payments p JOIN orders o ON p.order\_id = o.order\_id;

SELECT c.customer\_id, c.first\_name FROM customers c LEFT JOIN orders o ON c.customer\_id = o.customer\_id WHERE o.order\_id IS NULL;

SELECT p.product\_id, p.product\_name FROM products p LEFT JOIN order\_items oi ON p.product\_id = oi.product\_id WHERE oi.order\_id IS NULL;

SELECT o.customer\_id, c.first\_name, SUM(o.total\_amount) FROM orders o JOIN customers c ON o.customer\_id = c.customer\_id GROUP BY o.customer\_id, c.first\_name;

SELECT o.customer\_id, c.first\_name, SUM(oi.quantity) FROM orders o JOIN customers c ON o.customer\_id = c.customer\_id JOIN order\_items oi ON o.order\_id = oi.order\_id GROUP BY o.customer\_id, c.first\_name;

SELECT oi.order\_id, p.product\_name, oi.quantity FROM order\_items oi JOIN products p ON oi.product\_id = p.product\_id;

SELECT o.order\_id FROM orders o LEFT JOIN payments p ON o.order\_id = p.order\_id WHERE p.payment\_id IS NULL;

SELECT o.customer\_id, c.first\_name, MAX(o.order\_date) FROM orders o JOIN customers c ON o.customer\_id = c.customer\_id GROUP BY o.customer\_id, c.first\_name;

**4️⃣ Subqueries & Advanced Filters**

1️⃣ Find the **most expensive product** in the store using a subquery.  
2️⃣ Retrieve the list of **customers who have placed at least one order**.  
3️⃣ Display **orders where the total amount is greater than the average order amount**.  
4️⃣ Find the **cheapest product in each category** using a correlated subquery.  
5️⃣ Identify **the customer who has placed the highest number of orders**.  
6️⃣ Fetch the **second most expensive product** using an alternative ranking method.  
7️⃣ List all **customers who have never made a payment for any order**.  
8️⃣ Retrieve all **products with stock levels below the average stock quantity**.  
9️⃣ Find **customers who have spent more than $2000 in total on orders**.  
🔟 Identify employees who **earn the same salary as at least one other employee**.

SELECT \* FROM products WHERE price = (SELECT MAX(price) FROM products);

SELECT \* FROM customers WHERE customer\_id IN (SELECT DISTINCT customer\_id FROM orders);

SELECT \* FROM orders WHERE total\_amount > (SELECT AVG(total\_amount) FROM orders);

SELECT \* FROM products p WHERE price = (SELECT MIN(price) FROM products WHERE category = p.category);

SELECT customer\_id, COUNT(\*) FROM orders GROUP BY customer\_id HAVING COUNT(\*) = (SELECT MAX(order\_count) FROM (SELECT customer\_id, COUNT(\*) AS order\_count FROM orders GROUP BY customer\_id) AS order\_counts);

SELECT \* FROM products ORDER BY price DESC LIMIT 1 OFFSET 1;

SELECT \* FROM customers WHERE customer\_id NOT IN (SELECT DISTINCT customer\_id FROM orders JOIN payments USING(order\_id));

SELECT \* FROM products WHERE stock\_quantity < (SELECT AVG(stock\_quantity) FROM products);

SELECT customer\_id, SUM(total\_amount) FROM orders GROUP BY customer\_id HAVING SUM(total\_amount) > 2000;

SELECT \* FROM employees WHERE salary IN (SELECT salary FROM employees GROUP BY salary HAVING COUNT(\*) > 1);