Customer Orders Management System

This article provides a step-by-step guide to understanding how a retail order management system can store, manage, and analyze customer and sales data efficiently. The project is designed to give hands-on experience with core database concepts including :

- Creating structured relational databases for retail operations.
- Inserting and managing customer and order transaction data.
- Running SQL queries to uncover sales and customer behavior trends.
- Measuring revenue, product demand, and customer spending patterns.

By the end of this project, readers will understand how SQL can be used to manage and analyze retail data to support data-driven decisions.

Project Overview

This project simulates a simple retail store's order management system using SQL. It involves creating and managing customer and order data through relational tables. Using structured queries, the project analyzes purchase behavior, product performance, and revenue trends. The goal is to understand how SQL helps derive actionable insights from sales data in a real-world business context.

Table of Contents:

Let's start building !

Step 1: Setting Up the Database Schema

Create Database

```
create database project;
use project;
```

Customers Table

```
create table customers(
customerID INT Primary Key,
FirstName Varchar(50),
LastName varchar(50),
City Varchar(50),
JoinDate date
 );
```

Orders Table

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustomerID INT FOREIGN KEY REFERENCES Customers(CustomerID),
    OrderDate DATE,
    Product VARCHAR(50),
    Quantity INT,
    Price INT
);
```

Step 2: Data Insertion

After setting up the database schema, we insert sample data into the Customers and Orders tables. This data includes customer profiles from different cities and various product orders with varying quantities and prices. It provides a realistic dataset to perform meaningful analysis and generate business insights through SQL queries.

Insert Customers:

```
insert into customers
values (1,'John','Doe','Mumbai','2024-01-05'),
(2,'Alice','Smith','Delhi','2024-02-15'),
(3, 'Bob', 'Brown', 'Bangalore', '2024-03-20'),
(4, 'Sara', 'White', 'Mumbai', '2024-01-25'),
(5, 'Mike', 'Black', 'Chennai', '2024-02-10');
```

Insert Orders:

```
INSERT INTO Orders VALUES
(101, 1, '2024-04-10', 'Laptop', 1, 55000),
(102, 2, '2024-04-12', 'Mouse', 2, 800),
(103, 1, '2024-04-15', 'Keyboard', 1, 1500),
(104, 3, '2024-04-20', 'Laptop', 1, 50000),
(105, 4, '2024-04-22', 'Headphones', 1, 2000),
(106, 2, '2024-04-25', 'Laptop', 1, 52000),
(107, 5, '2024-04-28', 'Mouse', 1, 700),
(108, 3,'2024-05-02','Keyboard',1,1500);
```

Step 3 : SQL Queries to Explore Insights

Now that the database is populated with records, we will use SQL queries to answer real-world business questions. These queries aim to:

- Analyze customer purchase behavior
- Identify top-selling products
- Measure overall and monthly revenue
- Compare customer activity across cities

The next section includes SQL queries grouped by key business objectives:

- Customer Purchase Analysis
- Product Demand Trends
- Revenue & Sales Insights
- City-wise Customer Distribution
- High-Value Customer Identification

Step 4 : Business Analysis & Use Cases

1. **Get the list of all customers from Mumbai**

```
SELECT * FROM customers WHERE City = 'Mumbai';
```

Insight: John Doe and Sara White live in Mumbai.

2. Show all orders for Laptops

```
SELECT * FROM Orders WHERE Product = 'Laptop';
```

Insight: 3 laptops were sold: John, Bob, and Alice each purchased one.

3. Find the total number of orders placed

```
SELECT COUNT(*) AS TotalOrders FROM Orders;
```

Insight: A total of 8 orders were placed by all customers.

4. Find orders where price is between 50,000 and 80,000

```
SELECT * FROM Orders WHERE
Price BETWEEN 50000 AND
80000;
```

Insight: Orders by John, Bob, and Alice are in the high-value range of ₹50k–₹80k.

5. Full name of customers and the product they ordered

```
SELECT c.FirstName + ' ' +
c.LastName AS FullName,
o.Product
FROM customers c
JOIN Orders o ON
c.customerID =o.customerID;
```

Insight: John Doe ordered a Laptop and a Keyboard. Alice Smith bought both a Laptop and a Mouse.

6. Customers who have not placed any orders

```
SELECT * FROM customers
WHERE  customerID  NOT  IN
(SELECT DISTINCT customerID
FROM Orders);
```

Insight: All customers have placed at least one order.

7. Total revenue earned

```
SELECT SUM(quantity * price) AS total_revenue FROM Orders;
```

Insight: The company earned ₹1,63,000 from total orders.

8. Total quantity of Mouses sold

```
SELECT SUM(quantity) AS TotalMouseQuantity FROM Orders WHERE
Product = 'Mouse';
```

Insight: 3 Mouse units were sold — 2 by Alice and 1 by Mike.

9. Total sales per customer

```
SELECT c.FirstName, SUM(o.quantity * o.price)
FROM customers c
JOIN Orders o ON c.customerID = o.CustomerID
GROUP BY c.FirstName;
```

Insight: John is the top spender with ₹56,500 in purchases.

10. Number of orders per city

```
SELECT c.City, COUNT(o.OrderID) AS Orders
FROM customers c
JOIN Orders o ON c.customerID = o.CustomerID
GROUP BY c.City;
```

Insight: Mumbai leads with the highest number of orders (3).

11. Customers who spent more than ₹50,000

```
SELECT c.* FROM customers c
WHERE c.customerID IN (
SELECT CustomerID FROM Orders GROUP BY CustomerID HAVING SUM(price) >
50000);
```

Insight: John, Alice, and Bob each spent more than ₹50,000.

12. Label orders as 'High Value' or 'Low Value'

```
SELECT OrderID, Price,
CASE
WHEN Price > 50000 THEN 'High Value'
ELSE 'Low Value'
END AS ValueLabel
FROM Orders;
```

Insight: 3 orders are High Value (₹>50000), and 5 are Low Value.

13. Running total of revenue by order date

```
SELECT OrderID, OrderDate, Price,
SUM(Price) OVER (ORDER BY OrderDate) AS RunningRevenue
FROM Orders;
```

Insight: The revenue reached ₹1,63,000 by the last order.

14. Row number per customer by order date

```
SELECT OrderID, CustomerID, OrderDate, Price,
ROW_NUMBER() OVER (PARTITION BY CustomerID ORDER BY OrderDate)
AS RowNum
FROM Orders;
```

Insight: John placed 2 orders, with Laptop first, then Keyboard.

Step 6 : Conclusion

This project effectively demonstrates how SQL can be used to manage and analyze sales data in a retail environment. Through structured queries, we explored customer behavior, product trends, and revenue performance using real data. We applied various SQL techniques including joins, subqueries, aggregations, and window functions to generate meaningful insights. The analysis identified top spenders, popular products like laptops, and high-revenue customers such as John, Alice, and Bob. Overall, this project shows how database-driven insights can support business decision-making and strategic planning in retail.