🛒 Retail Data Analysis

This article provides a step-by-step guide to understanding how a **retail system** efficiently stores, manages, and analyzes operational data. The project serves as a practical approach to learning fundamental database and backend concepts, including:

- 🏬 Creating structured relational databases for retail operations.
- 🧑‍💻 Inserting and managing **customer**, **product**, **order**, and **transaction** data.
- 📊 Running SQL queries to reveal **business insights**.
- 🔍 Measuring **customer behaviour**, **product performance**, and **sales trends**.

By the end of this project, readers will understand how **SQL** is used to manage and analyse **retail business data** effectively.

📌 Project Overview

This case study simulates a retail company's database system. The goal is to understand how structured data can be used to store, manage, and analyze real-world retail operations. You'll create database tables, insert sample records, and write SQL queries to generate insights related to customer behavior, product performance, and revenue trends.

Pavan-M

Table of Contents:

Let's start building!

Step fi: Setting Up the Database Schema

Create Database

```
Create database project
use project;
```

Customers Table

```
 Create Table Customer (
     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 Customer(CustomerID),
     OrderDate Date,
     Product Varchar(50),
     Quantity INT,
     Price INT
     );
```

Step 2: Data Insertion

After setting up the schema, we now populate each table with initial sample data. This step ensures that we have enough variety to run real-time analytics and answer business queries later in the project.

Insert Customers:

```
Insert into Customer values
(1,'John','Doe','Mumbai','2024-01-05'),
(2,'Alice','White','Delhi','2024-02-15'),
(3,'Bob','Smith','Bangalore','2024-03-20'),
(4,'Sara','Brwon','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',1,1600);
```

Step 3 : SQL Queries to Explore Insights

Now that the database is set up with records, we'll begin answering real-world questions through SQL queries. These queries will help:

- Monitor sales efficiency and inventory management
- Understand customer buying patterns and preferences
- Track product performance and category trends
- Measure revenue growth and profitability across regions

The following section presents SQL queries organized by business objectives, along with explanations:

- Customer & Purchase Analysis
- Product Performance and Trends
- Revenue and Sales Trends
- Operational Efficiency and Stock Management
- Predictive Analytics and Comparative Metrics

Step 4 : Business Analysis & Use Cases

Part A: Basic Queries

1. Get the list of all customers from Mumbai

```
Select * from Customer where City = 'Mumbai';
```

Insight: Sara Ali and John Doe are from Mumbai, indicating the city has multiple active buyers.

2. Show all orders for Laptops.

```
Select * from Orders where Product = 'Laptop';
```

Insight: 3 Laptop orders were placed, highlighting high-value electronics demand.

3. Find the total number of orders placed

```
Select count(*)AS TotalOrders from Orders;
```

Insight: Total of 8 orders have been placed, showing moderate activity across the dataset.

🔷 Part B: Joins

4: Get the full name of customers and their products ordered

```
Select c.Firstname,+' '+
c.LastName AS FullName,
o.Product
from Customer c
Join Orders o ON
c.CustomerID=o.CustomerID;
```

Insight: Shows customer-product relationships, helpful for understanding preferences.

5.Find customers who have not placed any orders

```
Select c.CustomerID,o.orderid
from Customer c
Join Orders o ON c.CustomerID = o.CustomerID
Where o.OrderID is NULL
```

I

Insight: Rita Sharma has not placed any orders yet—potential target for marketing.

Pavan-M

⯁ Part C: Aggregations

6. Find the total revenue earned from all orders.

```
SELECT SUM(Price * Quantity) AS TotalRevenue
FROM Orders
JOIN Products ON Orders.ProductID = Products.ProductID;
```

Insight: Total revenue indicates the financial health of the retail system.

7: Find total quantity of Mouses sold.

```
Select sum(Quantity)AS TotalMouse from Orders
Where Product = 'Mouse';
```

Insight: A total of 5 mouse units sold, suggesting popular accessory item.

8. Show total sales amount per customer

```
Select c.Firstname, sum(o.quantity * o.Price)As sales
from Customer c Join Orders o on c.CustomerID = o.CustomerID
group by c.Firstname;
```

Insight: Identifies top spenders—useful for loyalty programs.

9. Show number of orders per city

```
Select c.city, count(o.orderid) as orders
From Customer c join Orders o ON c.CustomerID = o.CustomerID
Group by c.City;
```

Insight: Mumbai and Delhi lead in order volumes, indicating regional trends.

Pavan-M

◆ Part D: Sub-query & CASE

10. Find customer who spent more than 50000 in total

```
Select C.* from customer C
Where c.CustomerID IN (select CustomerID from Orders
Group by CustomerID
Having sum(price) > 50000 );
```

Insight: These are high-value customers ideal for premium offers.

11. Label orders as 'High Value' if price > ₹50,000:

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

Insight: Helps categorize transactions and strategize accordingly.

◆ Part E: Window Functions

12. Find the running total of revenue by order date.

```
SELECT OrderID, Orderdate, Price,
     sum(price) OVER (ORDER BY orderdate )AS RunningRevenue
from Orders;
```

Insight: Helps track cumulative growth over time.

Pavan-M

13: Assign a Row_Number to each order by CustomerID ordered by

```
select
     CustomerID,
     OrderID,
     Price,
     ROW_NUMBER() Over (PARTITION BY CustomerID ORDER BY Price) AS
OrderPrice
From Orders;
```

Orderdate(oldest first).

Insight: Tracks customer purchase sequence—useful for customer journey analysis.

14: Use RANK to rank orders by Price (Highest to Lowest)

```
Select
     OrderID,
     CustomerID,
     Price,
       RANK() OVER (ORDER BY Price DESC) AS PriceRank
From Orders;
```

Insight: Highlights top-expense orders for better cost analysis.

15: 15. Dense Rank on price (explain difference with RANK):

```
UPDATE Orders
Set Price = 52000
Where Price = 50000;

Select
     OrderID,
     CustomerID,
     Price,
DENSE_RANK() OVER (ORDER BY Price DESC) AS price_rank
FROM Orders;
```

Insight: Unlike `RANK`, `DENSE_RANK` avoids gaps in ranking when ties occur.

Pavan-M

16: Find customers who have placed more than 1 order using HAVING.

```
SELECT
    CustomerID,
    COUNT(OrderID) AS TotalOrders
FROM Orders
GROUP BY CustomerID
HAVING COUNT(OrderID) > 1;
```

Insight: These are repeat buyers—key for customer retention strategies.

Step 6: Conclusion

This project demonstrated the power of **SQL** in transforming **retail business data** into actionable insights. From **customer behavior** and **product performance** to **revenue trends** and **operational efficiency**, each query revealed useful patterns that help:

🎯 Optimize sales strategies

📈 Improve inventory planning

💼 Personalize customer engagement

💰 Boost profitability

A well-structured SQL database empowers data-driven decisions and fosters growth in the retail sector.