CASE STUDY #1

RAJESH'S RETAIL STORE

Data-Driven Growth in E-Commerce

Tabhine: *rajeshnikam.com*

1 Introduction

Objective

The objective of this case study to perform end-to-end data analysis on an e-commerce retail dataset using SQL, with the objective of generating business insights across customer behaviour, product performance, order trends, and return patterns. The goal is to build a strong foundation in querying real-world relational databases by writing basic, intermediate, and advanced SQL queries for business decision-making, KPI tracking, and dashboard creation.

1.1 Database Overview

This case study is based on five datasets:

1. **Customers (500 records)** – Contains customer details such as name, email, gender, age, country, and signup date.
2. **Products (50 records)** – Includes product details such as name, category, brand, and price.
3. **Orders (1,200 records)** – Stores order transactions, including customer purchases, total amount, order status, and payment method.
4. **Order Details (1,500 records)** – Contains product-level details of each order, including quantity and unit price.
5. **Returns (150 records)** – Stores return information including reason, status, and date of return.

These datasets are linked through:

- **CustomerID** (to associate orders with customers)
- **OrderID** (to connect orders with order details and returns)
- **ProductID** (to associate order details with products)

2. Database Schema Design:

2.1 Customers Table :-

Column Name	Data Type	Description
CustomerID	INT	Unique identifier for customer
Name	TEXT	Full name of the customer
Email	TEXT	Customer's email address
Gender	TEXT	Male / Female
Age	INT	Age of the customer
Country	TEXT	Country of residence
SignupDate	DATE	Date when customer registered

2.2 Orders Table :-

Column Name	Data Type	Description
OrderID	INT	Unique identifier for the order
Customer ID	INT	References customers_500
Order Date	DATE	Date the order was placed
Total Amount	DECIMAL	Total value of the order
Payment Method	TEXT	UPI / Credit Card / Net Banking etc.
Order Status	TEXT	Delivered / Returned / Cancelled

2.3 Order_details Table

Column Name	Data Type	Description
OrderDetailID	INT	Unique identifier for the line item
OrderID	INT	References orders_expanded
ProductID	INT	References products_expanded
Quantity	INT	Quantity of product ordered
UnitPrice	DECIMAL	Price per unit at the time of order

2.4 Products Table

Column Name	Data Type	Description
ProductID	INT	Unique product ID
ProductName	TEXT	Name of the product
Category	TEXT	Electronics / Beauty / Fashion
Brand	TEXT	Brand name
Price	INT	Retail price of the product

2.5 Returns Table

Column Name	Data Type	Description
ReturnID	INT	Unique return ID
OrderID	INT	References orders_expanded
ReturnDate	DATE	Date return was made
Reason	TEXT	Reason for return (Damaged, Late, etc.)
Status	TEXT	Return status (Approved / Pending / Rejected)

3.Data Processing:-

- Check for **null or missing values** and handle them appropriately.
- Validate **data types** to match the expected format.
- Ensure **foreign key integrity** between tables.
- Remove **duplicate records** if any exist.
- **Normalize data** to improve efficiency and avoid redundancy.
- Removed duplicate rows using DISTINCT or row ID checks.

----1.Show the first 10 customers and their countries.

```
SELECT
        CustomerID ,
        Name , Country
FROM
        customers
        LIMIT 10;
```

----2.List all customers who signed up in 2024.

```
SELECT
      *
FROM
      Customers
WHERE
      YEAR(SignupDate) = 2024;
```

----3.Count the total number of orders Shipped.

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

----4.Display all products under the “Beauty” category.

```
SELECT
      *
FROM
      products
WHERE
      Category = 'Beauty';
```

----5.Find all unique countries from the customers table.

```
SELECT
        DISTINCT Country
FROM
        customers;
```

----6.Show orders where total amount is exactly 5000

```
SELECT
      *
FROM
     Orders

WHERE

      TotalAmount = 5000;
```

----7.List customers older than 50.

```
SELECT
       *
FROM
        Customers

WHERE

        Age > 50;
```

----8.Display the most recent order from the orders table.

```
SELECT
        *
FROM
        orders
ORDER BY

        OrderDate DESC

         LIMIT 1;
```

----9.Count the number of Gmail users.

```
SELECT
    COUNT(*) AS GmailUsers
FROM
    customers
WHERE
     Email LIKE '%@gmail.com';
```

----10.Show customers whose name starts with 'A'.

```
SELECT
    *
FROM
    customers
WHERE
     Name LIKE 'A%';
```

----11.Display products with price above 5000.

```
SELECT
    *
FROM
    products
WHERE
    Price > 5000;
```

----12.List all male customers from Australia.

```
SELECT
    *
FROM
     customers
WHERE
     Gender = 'Male' AND
     Country = 'Australia';
```

----13.Show return records with status ‘Pending’.

```
SELECT
	*
FROM
	returns
WHERE
	Status = 'Pending';
```

----14.List all payment methods that include the word "Card"

```
SELECT
	DISTINCT PaymentMethod
FROM
	orders
WHERE
	PaymentMethod LIKE '%Card%';
```

----15.Count customers with age between 25 and 35

```
SELECT
	COUNT(*) AS Age_25_to_35
FROM
	customers
WHERE
	Age BETWEEN 25 AND 35;
```

INTERMEDIATE LEVEL

----16.List the top 5 customers based on total order value.

```
SELECT
    o.CustomerID, c.Name, SUM(o.TotalAmount) AS TotalSpent
FROM
    orders o
JOIN
    customers c ON o.CustomerID = c.CustomerID
GROUP BY
    o.CustomerID, c.Name
ORDER BY
    TotalSpent DESC
    LIMIT 5;
```

----17.Find the number of orders each customer placed.

```
SELECT
    CustomerID, COUNT(*) AS OrderCount
FROM
    orders
GROUP BY
    CustomerID;
```

----18.Show average age by country.

```
SELECT
    Country, AVG(Age) AS AverageAge
FROM
    customers
GROUP BY
    Country;
```

----19.List monthly return counts for the past 6 months.

```
SELECT
        DATE_FORMAT(ReturnDate, '%Y-%m') AS Month, COUNT(*)
        AS  ReturnCount
FROM
        Returns
WHERE
        ReturnDate >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH)
GROUP BY Month
ORDER BY Month;
```

#20.Display total revenue per product.

```
SELECT
         p.ProductID, p.ProductName, SUM(od.Quantity * od.UnitPrice)
        AS Revenue
FROM
         order_details od
JOIN
        products p ON od.ProductID = p.ProductID
GROUP BY
        p.ProductID, p.ProductName;
```

#21.Count total orders per brand.

```
SELECT
         p.Brand, COUNT(DISTINCT od.OrderID) AS OrderCount
FROM
        order_details od
JOIN
        products p ON od.ProductID = p.ProductID
GROUP BY
        p.Brand;
```

#22.Identify customers who made more than 2 returns

```
SELECT
        o.CustomerID, COUNT(*) AS ReturnCount
FROM
        returns r
JOIN
       orders o ON r.OrderID = o.OrderID
GROUP BY
       o.CustomerID
HAVING
       COUNT(*) > 2;
```

#23.Calculate average order value per month.

```
SELECT
       DATE_FORMAT(OrderDate, '%Y-%m') AS Month, AVG(TotalAmount)
       AS AvgOrderValue
FROM
        orders
GROUP BY
        Month
ORDER BY
        Month;
```

#24.Find the number of categories each brand operates in.

```
SELECT
         Brand, COUNT(DISTINCT Category) AS CategoryCount
FROM
         products
GROUP BY
         Brand;
```

----25.Show customers who placed orders using more than one payment method

```
SELECT
        CustomerID
FROM
       orders
GROUP BY
        CustomerID
HAVING
        COUNT(DISTINCT PaymentMethod) > 1;
```

26.Identify products never returned.

```
SELECT
        DISTINCT p.ProductID, p.ProductName
FROM
        products p
WHERE
        p.ProductID NOT IN (
                        SELECT od.ProductID
                        FROM order_details od
                        JOIN returns r ON od.OrderID = r.OrderID
);
```

----27.Show category-wise product count and total revenue.

```
SELECT
        p.Category, COUNT(DISTINCT p.ProductID) AS ProductCount,
         SUM(od.Quantity * od.UnitPrice) AS TotalRevenue
FROM
        products p
JOIN
        order_details od ON p.ProductID = od.ProductID
GROUP BY
       p.Category;
```

----28.Display orders that were both placed and returned.

```
SELECT
        o.*
FROM
        orders o
JOIN
        returns r ON o.OrderID = r.OrderID;
```

----29. List the number of unique products purchased by each customer.

```
SELECT
        o.CustomerID, COUNT(DISTINCT od.ProductID) AS UniqueProducts
FROM
       orders o
JOIN
       order_details od ON o.OrderID = od.OrderID
GROUP BY
       o.CustomerID;
```

----30.Show how many orders were placed each day last week.

```
SELECT
        OrderDate, COUNT(*) AS Orders
FROM
       orders
WHERE
       OrderDate >= DATE_SUB(CURDATE(), INTERVAL 7 DAY)
GROUP BY
      OrderDate;
```

----31.Find the top 3 categories with the most returns.

```
SELECT
        Category, ReturnCount
FROM
        (
        SELECT
        p.Category,
        COUNT(*) AS ReturnCount,
        RANK() OVER (ORDER BY COUNT(*) DESC) AS rnk
    FROM returns r
    JOIN order_details od ON r.OrderID = od.OrderID
    JOIN products p ON od.ProductID = p.ProductID
    GROUP BY p.Category
) ranked_returns
WHERE rnk <= 3;
```

----32.Identify customers who placed orders in at least 3 different months.

```
SELECT
        CustomerID
FROM
        orders
GROUP BY
        CustomerID
HAVING
        COUNT(DISTINCT DATE_FORMAT(OrderDate, '%Y-%m')) >= 3;
```

ADVANCED LEVEL

----33.Show product names and count of returns for each.

```
SELECT
        p.ProductName, COUNT(*) AS ReturnCount
FROM
        returns r
JOIN
       order_details od ON r.OrderID = od.OrderID
JOIN
      products p ON od.ProductID = p.ProductID
GROUP BY
      p.ProductName;
```

----34.Calculate the average time (in days) between signup and first order.

```
SELECT
      AVG(DaysToFirstOrder) AS AvgDaysToFirstOrder
FROM (
       SELECT c.CustomerID,
        DATEDIFF(MIN(o.OrderDate), c.SignupDate) AS DaysToFirstOrder
     FROM
           customers c
    JOIN
           orders o ON c.CustomerID = o.CustomerID
    GROUP BY
           c.CustomerID, c.SignupDate
) AS customer_first_order;
```

----**35.List all customers who returned at least 50% of their orders**.

```
SELECT
        o.CustomerID,
        COUNT(DISTINCT r.OrderID) * 1.0 / COUNT(DISTINCT o.OrderID)
         AS  ReturnRatio
FROM
        orders o
LEFT JOIN
         returns r ON o.OrderID = r.OrderID
GROUP BY
          o.CustomerID
HAVING
          ReturnRatio >= 0.5;
```

----**36.Calculate average revenue per user (ARPU) across countries.**

```
SELECT
          c.Country,
          ROUND(SUM(o.TotalAmount) / COUNT(DISTINCT c.CustomerID), 2)
          AS ARPU
FROM
          customers c
JOIN
          orders o ON c.CustomerID = o.CustomerID
GROUP BY
          c.Country;
```

----37. Identify top 5 customers with highest revenue but zero returns.

```
SELECT
        c.CustomerID,
        c.Name,
        SUM(o.TotalAmount) AS TotalRevenue
FROM
        customers c
JOIN
        orders o ON c.CustomerID = o.CustomerID
WHERE
        o.OrderID NOT IN (
    SELECT OrderID FROM returns
        )
GROUP BY
        c.CustomerID, c.Name
ORDER BY
        TotalRevenue DESC
LIMIT 5;
```

----38.Create a stored procedure GetCustomerSummary that takes a CustomerID as input and returns the total number of orders and total revenue.

```
DELIMITER //

CREATE PROCEDURE GetCustomerSummary(IN cid INT)

BEGIN

   SELECT

      CustomerID,

      COUNT(*) AS TotalOrders,

      SUM(TotalAmount) AS TotalRevenue

   FROM orders

   WHERE CustomerID = cid

   GROUP BY CustomerID;

END //


DELIMITER ;

CALL GetCustomerSummary(1001);
```

----39.Create a procedure TopSellingProducts that takes a number and returns top N products based on total quantity sold.

```
DELIMITER //

CREATE PROCEDURE TopSellingProducts(IN topN INT)
BEGIN
   SELECT
      p.ProductName,
      SUM(od.Quantity) AS TotalSold
   FROM order_details od
   JOIN products p ON od.ProductID = p.ProductID
   GROUP BY p.ProductID, p.ProductName
   ORDER BY TotalSold DESC
   LIMIT topN;
END //

DELIMITER ;

CALL TopSellingProducts(5);
```

----40.Write a procedure MonthlyRevenueReport that takes a year as input and shows revenue for each month of that year.

```
DELIMITER //

CREATE PROCEDURE MonthlyRevenueReport(IN inputYear INT)

BEGIN

   SELECT

      MONTH(OrderDate) AS Month,

      SUM(TotalAmount) AS MonthlyRevenue

   FROM orders

   WHERE YEAR(OrderDate) = inputYear

   GROUP BY MONTH(OrderDate)

   ORDER BY Month;

END //

DELIMITER ;


CALL MonthlyRevenueReport(2024);
```

----41.Create a procedure HighReturnCustomers that takes a number N and lists customers who have returned more than N orders.

```
DELIMITER //

CREATE PROCEDURE HighReturnCustomers(IN minReturns INT)

BEGIN

   SELECT

      c.CustomerID,

      c.Name,

      COUNT(r.ReturnID) AS ReturnCount

   FROM customers c

   JOIN orders o ON c.CustomerID = o.CustomerID

   JOIN returns r ON o.OrderID = r.OrderID

   GROUP BY c.CustomerID, c.Name

   HAVING ReturnCount > minReturns;

END //

DELIMITER ;

CALL HighReturnCustomers(2);
```

----42.Create a procedure ProductRevenueByDateRange that takes a start and end date and returns revenue per product.

```
DELIMITER //
CREATE PROCEDURE ProductRevenueByDateRange(IN startDate DATE, IN
endDate DATE)
BEGIN
   SELECT
      p.ProductName,
      SUM(od.Quantity * od.UnitPrice) AS TotalRevenue
   FROM orders o
   JOIN order_details od ON o.OrderID = od.OrderID
   JOIN products p ON od.ProductID = p.ProductID
   WHERE o.OrderDate BETWEEN startDate AND endDate
   GROUP BY p.ProductName
   ORDER BY TotalRevenue DESC;
END //
DELIMITER ;
CALL ProductRevenueByDateRange('2024-01-01', '2024-06-30');
```

5. Business Insights & Analysis

- **Customer Segmentation**

- Identified **high-value customers** who frequently purchase and generate the highest revenue.
- Segmented users by **age, country, and order frequency** for targeted marketing.

- **Product Performance**

- Ranked products by **total sales volume** and **revenue**.
- Flagged **underperforming items** with low sales and high return rates.

- **Sales Trends**

- Identified **peak order months**, with spikes during festive seasons and weekends.
- Analyzed trends in **payment methods**, showing increasing preference for UPI and Net Banking.

- **Inventory Management**

- Highlighted **top-selling items** likely to go out of stock.
- Flagged **slow-moving inventory** needing discounts or promotion.

- **Revenue Optimization**

- Found that **Electronics and Fashion** categories contribute over 60% of total revenue.
- Detected high **return rates** in some categories, affecting net profit margins.

---

6. Conclusion & Recommendations

6.1 Summary

This case study delivers a structured approach to analyzing an e-commerce business using SQL. By leveraging relational database concepts, we've uncovered key insights into customer behavior, order patterns, and product performance. These insights can guide **data-driven decision-making** to improve sales, reduce returns, and boost customer retention.

6.2 Recommendations

- **Target high-value customers** through personalized offers and loyalty rewards.
- **Restock high-demand products** more frequently to prevent lost sales.
- **Discount slow-moving items** to reduce holding costs.
- **Re-engage dormant customers** with win-back campaigns and email nudges.
- **Use sales seasonality** to better plan promotions and inventory cycles.

---

7. Future Scope

To deepen the analysis and generate real-time insights, the study can be extended by:

- **Predictive Analytics**
  Use machine learning to forecast customer lifetime value (CLV), product demand, or churn risk.
- **Real-Time Dashboards**
  Build live KPI dashboards using Power BI or Tableau for marketing, sales, and inventory monitoring.
- **Enhanced Dataset**
  Include more attributes like customer income, feedback ratings, or product reviews for **360° customer analysis**.