SQL Server Analytics Project: Customer Orders & Inventory

This project demonstrates practical SQL Server Management Studio (SSMS) usage for real-world business tasks involving customer orders, inventory management, and delivery performance.

# Data Overview

The dataset consists of the following tables imported from Excel into SQL Server:

- Customers-sql

- Products-sql

- Orders-sql

- Order\_Details-sql

- Deliveries-sql

# Step-by-Step Analysis & SQL Queries

## 1. Identify Customers with Missing Emails

Query:

SELECT \* FROM [dbo].[Customers-sql] WHERE Email IS NULL;

This detects customer records with missing email addresses for data cleaning.

## 2. Find Duplicate Customer Names

Query:

SELECT CustomerName, COUNT(\*) AS DuplicateCount FROM [dbo].[Customers-sql] GROUP BY CustomerName HAVING COUNT(\*) > 1;

Used to check for potential duplicate records.

## 3. Find Low Stock Products

Query:

SELECT ProductName, StockLevel FROM [dbo].[Products-sql] WHERE StockLevel < 20;

Helps inventory managers identify items needing restocking.

## 4. Delivery Performance by Month

Query:

SELECT FORMAT(d.DeliveryDate, 'yyyy-MM') AS DeliveryMonth, AVG(DATEDIFF(DAY, o.OrderDate, d.DeliveryDate)) AS AvgDeliveryDays FROM [dbo].[Orders-sql] o JOIN [dbo].[Deliveries-sql] d ON o.OrderID = d.OrderID WHERE d.DeliveryDate IS NOT NULL GROUP BY FORMAT(d.DeliveryDate, 'yyyy-MM');

Shows average delivery times per month.

## 5. Revenue by Product

Query:

SELECT p.ProductName, SUM(od.Quantity \* od.UnitPrice) AS TotalRevenue FROM [dbo].[Order\_Details-sql] od JOIN [dbo].[Products-sql] p ON od.ProductID = p.ProductID GROUP BY p.ProductName ORDER BY TotalRevenue DESC;

Calculates product revenue to identify best sellers.

## 6. Orders per Customer

Query:

SELECT c.CustomerName, COUNT(o.OrderID) AS OrderCount FROM [dbo].[Customers-sql] c JOIN [dbo].[Orders-sql] o ON c.CustomerID = o.CustomerID GROUP BY c.CustomerName ORDER BY OrderCount DESC;

Helps identify frequent customers.

## 7. Customer Segmentation

Query:

SELECT o.CustomerID, c.CustomerName, SUM(od.Quantity \* od.UnitPrice) AS TotalSpent, CASE WHEN SUM(od.Quantity \* od.UnitPrice) > 1000 THEN 'High' WHEN SUM(od.Quantity \* od.UnitPrice) BETWEEN 500 AND 1000 THEN 'Medium' ELSE 'Low' END AS CustomerSegment FROM [dbo].[Orders-sql] o JOIN [dbo].[Order\_Details-sql] od ON o.OrderID = od.OrderID JOIN [dbo].[Customers-sql] c ON o.CustomerID = c.CustomerID GROUP BY o.CustomerID, c.CustomerName ORDER BY TotalSpent DESC;

Classifies customers based on their spending.

## 8. Undelivered Orders

Query:

SELECT o.OrderID, c.CustomerName, p.ProductName FROM [dbo].[Orders-sql] o JOIN [dbo].[Order\_Details-sql] od ON o.OrderID = od.OrderID JOIN [dbo].[Products-sql] p ON od.ProductID = p.ProductID JOIN [dbo].[Customers-sql] c ON o.CustomerID = c.CustomerID LEFT JOIN [dbo].[Deliveries-sql] d ON o.OrderID = d.OrderID WHERE d.DeliveryDate IS NULL;

Shows pending or undelivered orders.

# Conclusion

This project demonstrates a complete business workflow using SQL — from cleaning and joining datasets, to generating operational and financial insights.