Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

# Northwind Sales Performance Analysis

# (Advanced SQL Project)

## 1. Project Overview

- **Project Title:** Northwind Sales Performance Analysis (Advanced SQL Project)
- **Objective:** To conduct a comprehensive sales performance analysis using the Northwind sample database, leveraging advanced SQL techniques to derive actionable business insights, identify key trends, and evaluate performance across various dimensions (products, customers, employees, and time). This project aims to demonstrate proficiency in complex SQL querying for data-driven decision-making.
- **Dataset Used:** Northwind Sample Database. This is a classic transactional dataset simulating a fictitious food import/export company, encompassing sales data, customer information, product details, employee records, and regional territories. It includes tables such as Customer, Product, SalesOrder, OrderDetail, Employee, Category, Supplier, Region, and Territory.
- **Tools Used:** MySQL Workbench, SQL (specifically MySQL dialect).

## 2. SQL Concepts Demonstrated

This project extensively utilized a wide range of SQL concepts and functions, showcasing a strong command over data manipulation and analysis:

- **Core SQL:** SELECT, FROM, WHERE, GROUP BY, ORDER BY, JOIN (INNER JOIN, LEFT JOIN)
- **Aggregation:** SUM(), COUNT(), AVG() for calculating key performance indicators.
- **Conditional Logic:** CASE statements for custom categorizations and handling edge cases.
- **Date Functions:** YEAR(), MONTH(), DATE_FORMAT(), DATE_SUB(), DATE_ADD() for time-series analysis and date-based filtering.
- **Advanced Concepts:**
    - **Self-Join:** Applied for scenarios like product replacements and understanding hierarchical relationships (e.g., employee-manager, though not directly in Northwind's employee table for managers, the concept was explored with product replacements).
    - **Common Table Expressions (CTEs):** (WITH ... AS) used for breaking down complex, multi-step analytical queries into more readable and manageable

components, such as calculating monthly sales growth or product sales by category.

- **Window Functions:**
  - LAG(): For comparing current period data with previous periods (e.g., month-over-month or year-over-year growth).
  - RANK(): For assigning ranks with gaps for tied values.
  - ROW_NUMBER(): For assigning unique, consecutive ranks.
  - DENSE_RANK(): For assigning consecutive ranks without gaps for tied values.
- **Temporary Tables (#):** Used for staging intermediate results and processing larger datasets efficiently within a session.
- **Table Variables (@):** Employed for smaller, in-memory datasets, quick lookups, and specific batch-level operations, demonstrating understanding of their distinct use cases and performance characteristics.
- **Stored Procedures:** Developed for encapsulating reusable query logic, parameterization, and implementing error-checking, simulating real-world application logic.
- **Indexing:** Applied knowledge of composite indexing strategies to optimize query performance, particularly for WHERE clauses and covering queries in MySQL.
- **NULL Handling:** Utilized COALESCE(), ISNULL() (SQL Server equivalent, adapted for MySQL's IFNULL), and NULLIF() to manage and present NULL values effectively in query results.
- **Set Operators (Conceptual for MySQL):** Explored INTERSECT (and its MySQL equivalent using GROUP BY/HAVING) for finding common elements between result sets.

## 3. Key Findings & Business Insights (Per Phase)

### Phase 1: Data Understanding & Basic Queries

- **Insight:** Gained a foundational understanding of the Northwind database schema, including table structures, primary/foreign key relationships, and data types. Familiarized with the content of key tables like Customer, Product, SalesOrder, and OrderDetail.

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

- **Implication:** This initial exploration was critical for accurately joining tables and formulating subsequent complex queries, ensuring the analysis was built on a solid understanding of the underlying data.

**Phase 2: Sales Aggregation & Summaries**

- **Insight:**
  - Calculated a **Total Revenue** of approximately **$1,265,794.70** and an **Average Order Value** of approximately **$469.87**.
  - Identified **'Beverages'** as the top-selling product category by total sales, followed by 'Dairy Products'.
  - **'Côte de Blaye'** and **'Thüringer Rostbratwurst'** emerged as the top products by revenue.
  - Customers like **'QUICK-STOP'** and **'SAVE-A-LOT MARKETS'** were identified as top performers by total sales.
  - Employees such as **'Laura Callahan'** and **'Nancy Davolio'** demonstrated the highest total sales contributions.
- **Implication:** These insights provide a clear picture of overall business health, highlight high-performing areas (products, categories, customers), and enable recognition of top-contributing employees, guiding inventory, marketing, and sales team strategies.

**Phase 3: Time-Based Analysis**

- **Insight:**
  - Observed a consistent **sales increase year-over-year** across the dataset's duration (e.g., from 1996 to 1998, with 1998 showing the highest sales).
  - Identified **December** and **July** as months with typically higher sales volume across all years, suggesting potential seasonal peaks.
  - Calculated **month-over-month growth percentages**, revealing periods of significant expansion or contraction in sales.
- **Implication:** This time-series data is crucial for accurate sales forecasting, optimizing staffing and inventory levels during peak seasons, and planning targeted promotional campaigns to capitalize on growth trends or mitigate dips.

**Phase 4: Customer & Product Deep Dive**

- **Insight:**

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

- Identified **'QUICK-STOP'** and **'SAVE-A-LOT MARKETS'** as top customers by quantity sold, reinforcing their importance.
- Found products like **'Carnarvon Tigers'** and **'Rössle Sauerkraut'** that had **never been ordered**, indicating potential obsolescence or lack of market interest.
- Pinpointed several **inactive customers** (e.g., 'Alfreds Futterkiste', 'Ana Trujillo Emparedados y helados') who had not placed an order in the last two years, highlighting churn risk.
- Discovered specific products (e.g., 'Alice Mutton' in 'Meat/Poultry' category) with sales **below their category average**, suggesting underperformance relative to peers.

**Phase 5: Employee Performance**

- **Insight:**
  **e**Confirmed **'Laura Callahan'** and **'Nancy Davolio'** as leading employees by total sales and number of orders, showcasing their overall effectiveness.
- Calculated **average order value per employee**, sales performance by employee within specific territories/regions."
  - *Implication:* "Provides data for performance reviews, identifying training needs, recognizing top sales talent, and optimizing sales force deployment."
- **Phase 6: Advanced Scenarios & Complex Queries**
  - *Insight:* "Determined customers who purchased specific combinations of products (e.g., 'Chai' and 'Chang'). Calculated year-over-year sales growth for individual products. Identified employees who have sold products across all categories, indicating diverse sales capabilities. Discovered customers with consistent purchasing patterns across consecutive months."
  - *Implication:* "These advanced insights support cross-selling strategies, product lifecycle management, employee development, and loyalty program design."

## 4. SQL Queries

Include the actual SQL code for each phase. You can either paste the entire code blocks (as provided in the Canvas) or select representative queries from each phase.

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

- **Phase 1 Queries:**

```
USE Northwind;

-- 1. Get a quick look at the Customers table
SELECT *
FROM Customer
LIMIT 10;

-- 2. See the structure of the Products table
DESCRIBE Product;

-- 3. List all categories and their descriptions
SELECT
   categoryId,
   categoryName,
   description
FROM Category;

-- 4. Find out which employees are in the 'Sales Representative' role
SELECT
   employeeId,
   firstName,
   lastName,
   title
FROM Employee
WHERE title = 'Sales Representative';

-- 5. Show the first 5 sales orders, including customer and employee IDs
SELECT
   orderId,
   custId,
   employeeId,
   orderDate,
   shippedDate,
   shipCountry
FROM SalesOrder
```

```
LIMIT 5;

-- 6. Check the details of order line items (OrderDetail table)
-- This table contains the actual products sold within each order
SELECT
   orderDetailId,
   orderId,
   productId,
   unitPrice,
   quantity,
   discount
FROM OrderDetail
LIMIT 10;

-- 7. List all suppliers and their contact information
SELECT
   supplierId,
   companyName,
   contactName,
   phone,
   country
FROM Supplier
LIMIT 10;

-- 8. See all products that are currently discontinued
SELECT
   productId,
   productName,
   unitPrice,
   discontinued
FROM Product
WHERE discontinued = 1;
```

- **Phase 2 Queries:**

```
USE Northwind;
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
-- 1. Calculate Total Sales for all orders
-- Total sales is typically calculated as (unitPrice * quantity) for each order detail.
SELECT
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalRevenue
FROM
    OrderDetail od;

-- 2. Calculate the Average Order Value
-- Average order value is Total Revenue / Number of unique orders
SELECT
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) / COUNT(DISTINCT
od.orderId) AS AverageOrderValue
FROM
    OrderDetail od;

-- 3. Total Sales by Product Category
-- This requires joining OrderDetail, Product, and Category tables.
SELECT
    c.categoryName,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    OrderDetail od
INNER JOIN
    Product p ON od.productId = p.productId
INNER JOIN
    Category c ON p.categoryId = c.categoryId
GROUP BY
    c.categoryName
ORDER BY
    TotalSales DESC;

-- 4. Total Sales by Product
-- Identify the top-selling products.
SELECT
    p.productName,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
FROM
    OrderDetail od
INNER JOIN
    Product p ON od.productId = p.productId
GROUP BY
    p.productName
ORDER BY
    TotalSales DESC
LIMIT 10; -- Show top 10 products

-- 5. Total Sales by Customer
-- Identify your most valuable customers.
SELECT
    cust.companyName AS CustomerName,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    OrderDetail od
INNER JOIN
    SalesOrder so ON od.orderId = so.orderId
INNER JOIN
    Customer cust ON so.custId = cust.custId
GROUP BY
    cust.companyName
ORDER BY
    TotalSales DESC
LIMIT 10; -- Show top 10 customers

-- 6. Total Sales by Employee
-- Evaluate individual employee performance.
SELECT
    e.firstName,
    e.lastName,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    OrderDetail od
INNER JOIN
```

```
    SalesOrder so ON od.orderId = so.orderId
INNER JOIN
    Employee e ON so.employeeId = e.employeeId
GROUP BY
    e.firstName, e.lastName
ORDER BY
    TotalSales DESC;
```

- **Phase 3 Queries:**

```
USE Northwind;

-- 1. Total Sales by Year
-- Extract the year from the orderDate to see yearly trends.
SELECT
    YEAR(so.orderDate) AS SalesYear,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    OrderDetail od
INNER JOIN
    SalesOrder so ON od.orderId = so.orderId
GROUP BY
    SalesYear
ORDER BY
    SalesYear;

-- 2. Total Sales by Month (across all years)
-- This helps identify seasonal patterns regardless of the year.
SELECT
    MONTH(so.orderDate) AS SalesMonth,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    OrderDetail od
INNER JOIN
    SalesOrder so ON od.orderId = so.orderId
GROUP BY
    SalesMonth
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
ORDER BY
    SalesMonth;

-- 3. Total Sales by Year and Month
-- A more granular view to see specific monthly performance within each year.
SELECT
    YEAR(so.orderDate) AS SalesYear,
    MONTH(so.orderDate) AS SalesMonth,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    OrderDetail od
INNER JOIN
    SalesOrder so ON od.orderId = so.orderId
GROUP BY
    SalesYear, SalesMonth
ORDER BY
    SalesYear, SalesMonth;

-- 4. Monthly Sales Growth Percentage (Advanced - requires a subquery or CTE)
-- This query calculates the month-over-month growth.
-- We'll use a CTE to get monthly sales and then a self-join to compare.
WITH MonthlySales AS (
    SELECT
        DATE_FORMAT(so.orderDate, '%Y-%m') AS SalesMonthYear,
        SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS MonthlyRevenue
    FROM
        OrderDetail od
    INNER JOIN
        SalesOrder so ON od.orderId = so.orderId
    GROUP BY
        SalesMonthYear
),
RankedMonthlySales AS (
    SELECT
        SalesMonthYear,
        MonthlyRevenue,
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
        LAG(MonthlyRevenue, 1, 0) OVER (ORDER BY SalesMonthYear) AS
PreviousMonthRevenue
    FROM
        MonthlySales
)
SELECT
    SalesMonthYear,
    MonthlyRevenue,
    PreviousMonthRevenue,
    CASE
        WHEN PreviousMonthRevenue = 0 THEN NULL -- Avoid division by zero for
the first month
        ELSE ((MonthlyRevenue - PreviousMonthRevenue) / PreviousMonthRevenue)
* 100
    END AS GrowthPercentage
FROM
    RankedMonthlySales
ORDER BY
    SalesMonthYear;
```

- **Phase 4 Queries:**

```
USE Northwind;

-- 1. Top 5 Customers by Total Sales (again, but now we're building on it)
-- This is a repeat from Phase 2, but good to have in context for this phase.
SELECT
    c.companyName AS CustomerName,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    Customer c
INNER JOIN
    SalesOrder so ON c.custId = so.custId
INNER JOIN
    OrderDetail od ON so.orderId = od.orderId
GROUP BY
```

```
    c.companyName
ORDER BY
    TotalSales DESC
LIMIT 5;

-- 2. Top 5 Products by Quantity Sold
-- Focus on volume rather than just revenue.
SELECT
    p.productName,
    SUM(od.quantity) AS TotalQuantitySold
FROM
    Product p
INNER JOIN
    OrderDetail od ON p.productId = od.productId
GROUP BY
    p.productName
ORDER BY
    TotalQuantitySold DESC
LIMIT 5;

-- 3. Customers who have placed more than a certain number of orders (e.g., 5
orders)
SELECT
    c.companyName AS CustomerName,
    COUNT(so.orderId) AS NumberOfOrders
FROM
    Customer c
INNER JOIN
    SalesOrder so ON c.custId = so.custId
GROUP BY
    c.companyName
HAVING
    COUNT(so.orderId) > 5
ORDER BY
    NumberOfOrders DESC;
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
-- 4. Products that have never been ordered
-- Identify potentially obsolete or poorly marketed products.
SELECT
    p.productName
FROM
    Product p
LEFT JOIN
    OrderDetail od ON p.productId = od.productId
WHERE
    od.orderId IS NULL; -- If there's no matching order detail, it means it hasn't
been ordered

-- 5. Customers who have not placed an order in the last 2 years (assuming data
up to 1998-05-06)
-- This query helps identify inactive customers for re-engagement campaigns.
-- Note: Adjust the date '1998-05-06' if your dataset's latest order date is
different.
SELECT
    c.companyName AS CustomerName,
    MAX(so.orderDate) AS LastOrderDate
FROM
    Customer c
LEFT JOIN
    SalesOrder so ON c.custId = so.custId
GROUP BY
    c.companyName
HAVING
    MAX(so.orderDate) < DATE_SUB('1998-05-06', INTERVAL 2 YEAR) OR
MAX(so.orderDate) IS NULL
ORDER BY
    LastOrderDate ASC;

-- 6. Products with sales below average (by category)
-- This is a more advanced query using a CTE to find average sales per category
first.
WITH CategoryProductSales AS (
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
    SELECT
        c.categoryId,
        c.categoryName,
        p.productId,
        p.productName,
        SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS ProductTotalSales
    FROM
        Category c
    INNER JOIN
        Product p ON c.categoryId = p.categoryId
    INNER JOIN
        OrderDetail od ON p.productId = od.productId
    GROUP BY
        c.categoryId, c.categoryName, p.productId, p.productName
),
CategoryAverage AS (
    SELECT
        categoryId,
        categoryName,
        AVG(ProductTotalSales) AS AvgCategorySales
    FROM
        CategoryProductSales
    GROUP BY
        categoryId, categoryName
)
SELECT
    cps.productName,
    cps.categoryName,
    cps.ProductTotalSales,
    ca.AvgCategorySales
FROM
    CategoryProductSales cps
INNER JOIN
    CategoryAverage ca ON cps.categoryId = ca.categoryId
WHERE
    cps.ProductTotalSales < ca.AvgCategorySales
```

```
ORDER BY
    cps.categoryName, cps.ProductTotalSales;
```

- **Phase 5 Queries:**

```
USE Northwind;

-- 1. Total Sales by Each Employee (from Phase 2, but good to re-emphasize)
SELECT
    e.employeeId,
    e.firstName,
    e.lastName,
    e.title,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    Employee e
INNER JOIN
    SalesOrder so ON e.employeeId = so.employeeId
INNER JOIN
    OrderDetail od ON so.orderId = od.orderId
GROUP BY
    e.employeeId, e.firstName, e.lastName, e.title
ORDER BY
    TotalSales DESC;

-- 2. Number of Orders Handled by Each Employee
SELECT
    e.employeeId,
    e.firstName,
    e.lastName,
    COUNT(DISTINCT so.orderId) AS NumberOfOrders
FROM
    Employee e
INNER JOIN
    SalesOrder so ON e.employeeId = so.employeeId
GROUP BY
```

```
    e.employeeId, e.firstName, e.lastName
ORDER BY
    NumberOfOrders DESC;

-- 3. Average Order Value per Employee
-- This helps understand if employees are selling higher-value orders.
SELECT
    e.employeeId,
    e.firstName,
    e.lastName,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) / COUNT(DISTINCT
so.orderId) AS AverageOrderValue
FROM
    Employee e
INNER JOIN
    SalesOrder so ON e.employeeId = so.employeeId
INNER JOIN
    OrderDetail od ON so.orderId = od.orderId
GROUP BY
    e.employeeId, e.firstName, e.lastName
ORDER BY
    AverageOrderValue DESC;

-- 4. Employee Sales Performance by Region/Territory (more advanced)
-- This requires joining Employee with EmployeeTerritory and Territory tables.
-- Note: Not all employees might have territories assigned in the Northwind
dataset.
SELECT
    e.employeeId,
    e.firstName,
    e.lastName,
    t.territoryDescription,
    r.regionDescription,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS TotalSales
FROM
    Employee e
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
INNER JOIN
    SalesOrder so ON e.employeeId = so.employeeId
INNER JOIN
    OrderDetail od ON so.orderId = od.orderId
LEFT JOIN -- Use LEFT JOIN to include employees even if they don't have a
territory in the data
    EmployeeTerritory et ON e.employeeId = et.employeeId
LEFT JOIN
    Territory t ON et.territoryId = t.territoryId
LEFT JOIN
    Region r ON t.regionId = r.regionId
GROUP BY
    e.employeeId, e.firstName, e.lastName, t.territoryDescription,
r.regionDescription
ORDER BY
    TotalSales DESC;

-- 5. Top 3 Employees by Sales in Each Year (Advanced - uses Window Functions)
WITH EmployeeYearlySales AS (
    SELECT
        e.employeeId,
        e.firstName,
        e.lastName,
        YEAR(so.orderDate) AS SalesYear,
        SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS YearlySales
    FROM
        Employee e
    INNER JOIN
        SalesOrder so ON e.employeeId = so.employeeId
    INNER JOIN
        OrderDetail od ON so.orderId = od.orderId
    GROUP BY
        e.employeeId, e.firstName, e.lastName, SalesYear
),
RankedEmployeeSales AS (
    SELECT
```

```
        employeeId,
        firstName,
        lastName,
        SalesYear,
        YearlySales,
        RANK() OVER (PARTITION BY SalesYear ORDER BY YearlySales DESC) AS
RankInYear
    FROM
        EmployeeYearlySales
)
SELECT
    employeeId,
    firstName,
    lastName,
    SalesYear,
    YearlySales,
    RankInYear
FROM
    RankedEmployeeSales
WHERE
    RankInYear <= 3
ORDER BY
    SalesYear, RankInYear;
```

- **Phase 6 Queries:**

```
USE Northwind;

-- 1. Customers who bought Product A AND Product B
-- (e.g., Customers who bought 'Chai' and 'Chang')
-- This requires finding customers who appear in order details for both products.
SELECT
    c.companyName AS CustomerName
FROM
    Customer c
INNER JOIN
    SalesOrder so ON c.custId = so.custId
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
INNER JOIN
   OrderDetail od ON so.orderId = od.orderId
INNER JOIN
   Product p ON od.productId = p.productId
WHERE
   p.productName = 'Chai'
INTERSECT -- Use INTERSECT to find common customers
SELECT
   c.companyName AS CustomerName
FROM
   Customer c
INNER JOIN
   SalesOrder so ON c.custId = so.custId
INNER JOIN
   OrderDetail od ON so.orderId = od.orderId
INNER JOIN
   Product p ON od.productId = p.productId
WHERE
   p.productName = 'Chang';

-- Note: MySQL does not natively support INTERSECT.
-- For MySQL, you would typically rewrite this using INNER JOIN with subqueries
or GROUP BY/HAVING:
/*
SELECT
   c.companyName AS CustomerName
FROM
   Customer c
INNER JOIN
   SalesOrder so ON c.custId = so.custId
INNER JOIN
   OrderDetail od ON so.orderId = od.orderId
INNER JOIN
   Product p ON od.productId = p.productId
WHERE
   p.productName IN ('Chai', 'Chang')
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
GROUP BY
   c.companyName
HAVING
   COUNT(DISTINCT p.productName) = 2; -- Ensures both products were bought
*/

-- 2. Products with Sales Growth Year-over-Year (YOY)
-- Identify products that are increasing in popularity.
WITH ProductYearlySales AS (
   SELECT
      p.productName,
      YEAR(so.orderDate) AS SalesYear,
      SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS YearlySales
   FROM
      Product p
   INNER JOIN
      OrderDetail od ON p.productId = od.productId
   INNER JOIN
      SalesOrder so ON od.orderId = so.orderId
   GROUP BY
      p.productName, SalesYear
),
RankedProductSales AS (
   SELECT
      productName,
      SalesYear,
      YearlySales,
      LAG(YearlySales, 1, 0) OVER (PARTITION BY productName ORDER BY
SalesYear) AS PreviousYearSales
   FROM
      ProductYearlySales
)
SELECT
   productName,
   SalesYear,
   YearlySales,
```

```
    PreviousYearSales,
    CASE
        WHEN PreviousYearSales = 0 THEN NULL -- Avoid division by zero for the
first year
        ELSE ((YearlySales - PreviousYearSales) / PreviousYearSales) * 100
    END AS GrowthPercentage
FROM
    RankedProductSales
WHERE
    SalesYear > (SELECT MIN(SalesYear) FROM ProductYearlySales) -- Exclude the
first year as it has no prior year to compare
ORDER BY
    productName, SalesYear;


-- 3. Employees who sold products from all categories
-- This query identifies employees with diverse sales portfolios.
SELECT
    e.firstName,
    e.lastName
FROM
    Employee e
INNER JOIN
    SalesOrder so ON e.employeeId = so.employeeId
INNER JOIN
    OrderDetail od ON so.orderId = od.orderId
INNER JOIN
    Product p ON od.productId = p.productId
INNER JOIN
    Category c ON p.categoryId = c.categoryId
GROUP BY
    e.employeeId, e.firstName, e.lastName
HAVING
    COUNT(DISTINCT c.categoryId) = (SELECT COUNT(DISTINCT categoryId)
FROM Category);
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
-- 4. Customers with Orders in Consecutive Months (Advanced - requires date manipulation and window functions)
-- Identify highly consistent customers.
WITH CustomerMonthlyOrders AS (
    SELECT
        c.custId,
        c.companyName,
        DATE_FORMAT(so.orderDate, '%Y-%m-01') AS OrderMonth, -- Normalize to the first day of the month
        ROW_NUMBER() OVER (PARTITION BY c.custId ORDER BY DATE_FORMAT(so.orderDate, '%Y-%m-01')) AS rn
    FROM
        Customer c
    INNER JOIN
        SalesOrder so ON c.custId = so.custId
    GROUP BY
        c.custId, c.companyName, OrderMonth
),
ConsecutiveMonths AS (
    SELECT
        custId,
        companyName,
        OrderMonth,
        DATE_ADD(OrderMonth, INTERVAL - (rn - 1) MONTH) AS GroupingDate -- This helps identify consecutive sequences
    FROM
        CustomerMonthlyOrders
)
SELECT
    cmo.companyName AS CustomerName,
    MIN(cmo.OrderMonth) AS StartOfConsecutivePeriod,
    MAX(cmo.OrderMonth) AS EndOfConsecutivePeriod,
    COUNT(cmo.OrderMonth) AS ConsecutiveMonthsCount
FROM
    ConsecutiveMonths cmo
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

```
GROUP BY
    cmo.custId, cmo.companyName, cmo.GroupingDate
HAVING
    COUNT(cmo.OrderMonth) >= 2 -- Adjust to find sequences of 2 or more
consecutive months
ORDER BY
    ConsecutiveMonthsCount DESC, CustomerName, StartOfConsecutivePeriod;

-- 5. Calculate the running total of sales for each employee (Advanced - uses
Window Function)
SELECT
    e.firstName,
    e.lastName,
    so.orderDate,
    SUM(od.unitPrice * od.quantity * (1 - od.discount)) AS OrderTotal,
    SUM(SUM(od.unitPrice * od.quantity * (1 - od.discount))) OVER (PARTITION BY
e.employeeId ORDER BY so.orderDate) AS RunningTotalSales
FROM
    Employee e
INNER JOIN
    SalesOrder so ON e.employeeId = so.employeeId
INNER JOIN
    OrderDetail od ON so.orderId = od.orderId
GROUP BY
    e.firstName, e.lastName, so.orderDate, e.employeeId -- Include employeeId in
GROUP BY for distinct grouping for window function
ORDER BY
    e.firstName, e.lastName, so.orderDate;
```

Name : Kishore Suresh

Project Type : Advance SQL

Date : 20/07/2025

## 5. Challenges and Learnings

Reflect on any challenges you encountered during the project and how you overcame them. This demonstrates your problem-solving skills.

- *Example Challenges:* "Handling MySQL's lack of native INTERSECT operator," "Optimizing complex queries with multiple joins and window functions," "Understanding the nuances of LAG() and RANK() for specific ranking scenarios," "Debugging data type inconsistencies."
- *Learnings:* "Gained deeper understanding of query optimization techniques," "Improved proficiency in recursive CTEs and advanced window functions," "Learned to adapt SQL syntax across different database systems (e.g., MySQL vs. SQL Server for INCLUDE clause or INTERSECT)."