Project 1 - Team 3 - Hasnatul Hosna

Top 3 Best Problems

-- 1

Query 1 : Proposition: Analyze sales performance by employee, grouping orders by employee and summarizing **total orders** and **total freight** cost to provide insights into each employee's sales activity. Retrieve the employee information such as full name, title, city, and country for a comprehensive analysis.

Why is this a top problem?

Efficiency:

It uses Common Table Expressions (CTEs) to organize the query logically and improve readability without sacrificing performance.

The query retrieves the necessary data using straightforward aggregation functions like COUNT() and SUM() without unnecessary complexity.

Readability:

The query is well-structured and easy to understand, making it accessible for both developers and analysts.

Meaningful aliases ('o' for Sales.Order, 'e' for HumanResources.Employee) are used, enhancing clarity.

Maintainability:

The use of CTEs separates logical sections of the query, making it easier to maintain and modify in the future.

Column aliases and descriptive comments could be added for further clarity, but even without them, the query is relatively self-explanatory.

In [1]:
```
use Northwinds2022TSQLV7;

WITH SalesSummary AS (
    SELECT
        o.EmployeeId,
        COUNT(o.OrderId) AS TotalOrders,
        SUM(o.Freight) AS TotalFreight
    FROM
        Sales.[Order] o
    GROUP BY
        o.EmployeeId
),
EmployeeInfo AS (
    SELECT
        e.EmployeeId,
        CONCAT(e.EmployeeFirstName, ' ', e.EmployeeLastName) AS FullName,
        e.EmployeeTitle,
        e.EmployeeCity,
```

```
        e.EmployeeCountry
    FROM
        HumanResources.Employee e
)
SELECT
    e.EmployeeId,
    e.FullName,
    e.EmployeeTitle,
    e.EmployeeCity,
    e.EmployeeCountry,
    s.TotalOrders,
    s.TotalFreight
FROM
    EmployeeInfo e
JOIN
    SalesSummary s ON e.EmployeeId = s.EmployeeId;
```

(9 rows affected)

Total execution time: 00:00:00.315

Out[1]:

EmployeeId	FullName	EmployeeTitle	EmployeeCity	EmployeeCountry	TotalOrders	TotalFreight
9	Patricia Doyle	Sales Representative	London	UK	43	3326.26
3	Judy Lew	Sales Manager	Kirkland	USA	127	10884.74
6	Paul Suurs	Sales Representative	London	UK	67	3780.47
7	Russell King	Sales Representative	London	UK	72	6665.44
1	Sara Davis	CEO	Seattle	USA	123	8836.64
4	Yael Peled	Sales Representative	Redmond	USA	156	11346.14
5	Sven Mortensen	Sales Manager	London	UK	42	3918.71
2	Don Funk	Vice President, Sales	Tacoma	USA	96	8696.41
8	Maria Cameron	Sales Representative	Seattle	USA	104	7487.88

Subsystem of `Northwinds2022TSQLV7`:

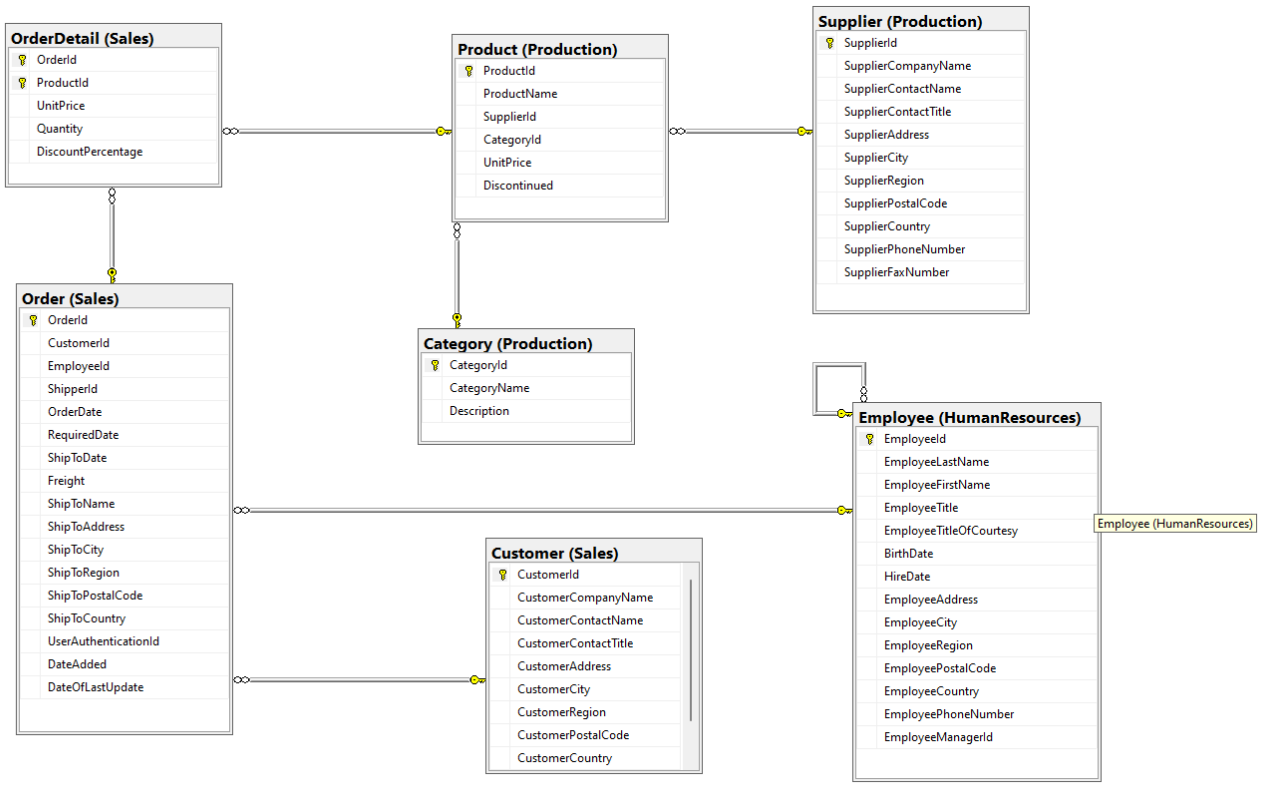
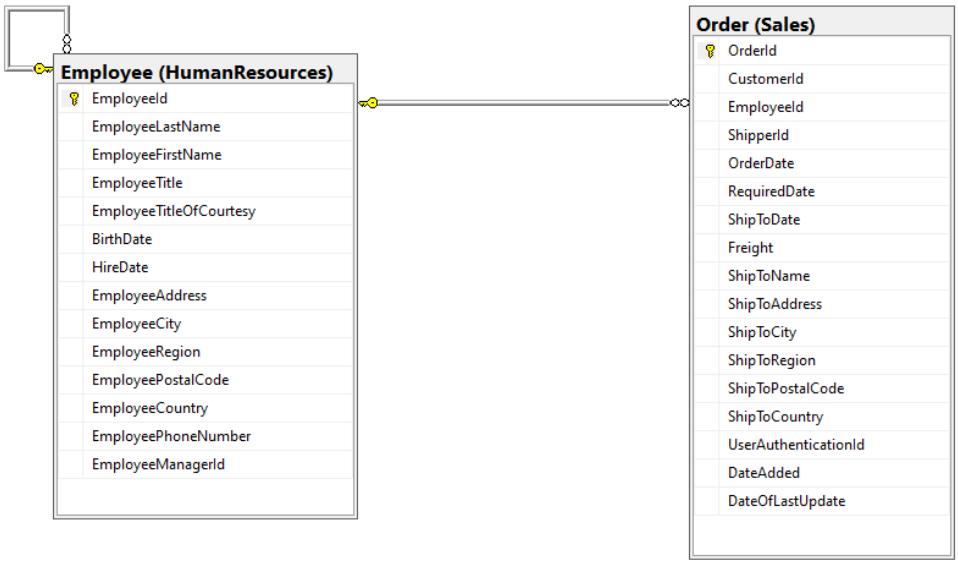
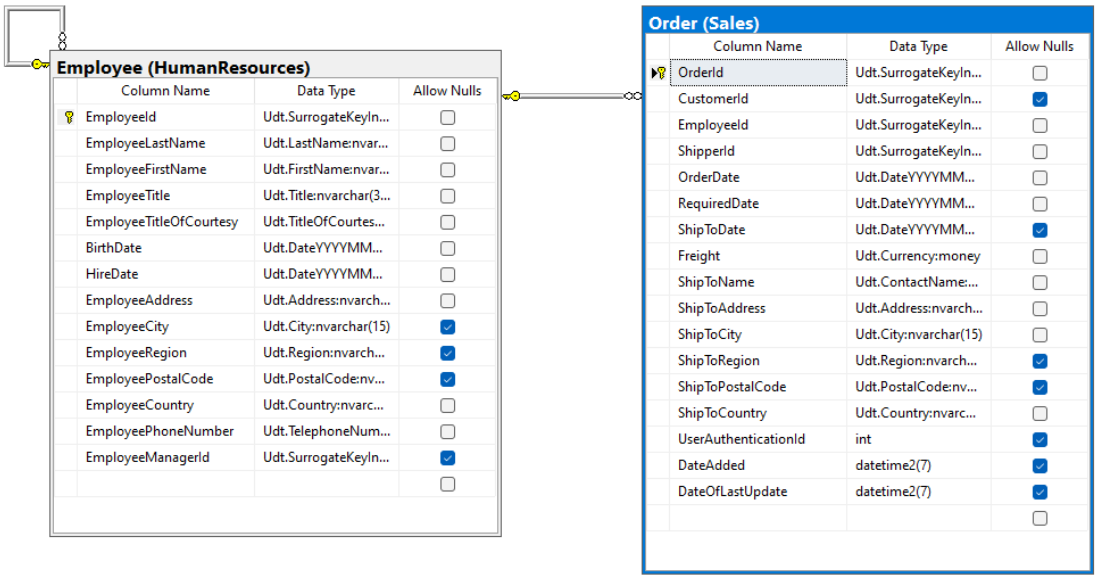


Diagram of Tables:



Column Standard of Tables:



--3

Query 2:

Proposition: Retrieves the total number of orders and the total sales amount for each customer company.

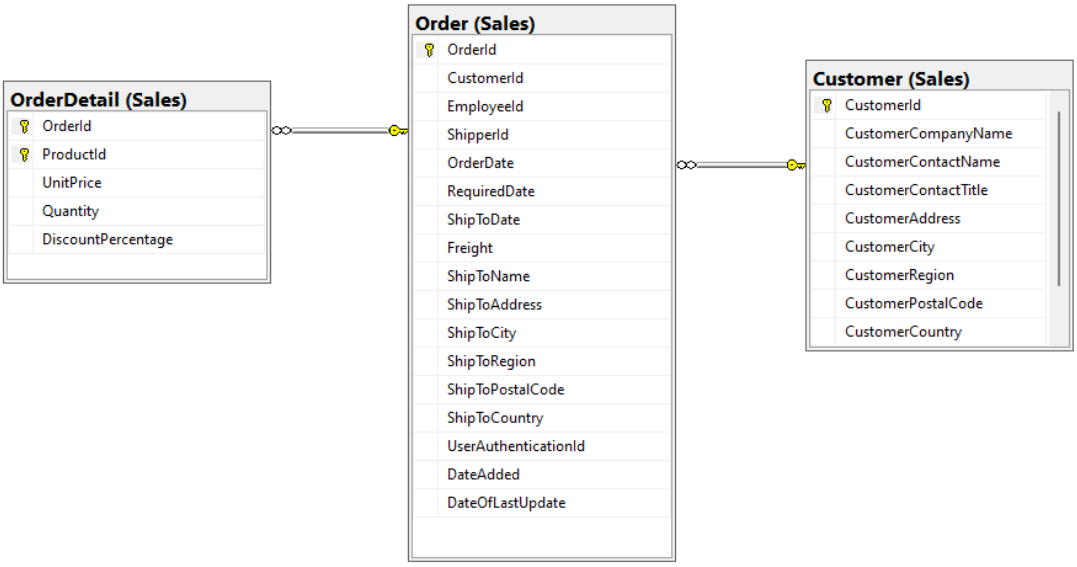
Why is this a top problem?

This query stands out as one of the best because of its efficiency and clarity in summarizing sales data for each customer. By utilizing common table expressions (CTEs) and aggregating functions, it elegantly computes the total number of orders and the total sales amount for each customer company. Firstly, the query uses a CTE named "OrderSummary" to calculate the total amount for each order by multiplying the unit price, quantity, and discount percentage. This CTE enhances readability and simplifies the main query by abstracting complex computations into a named subquery. Secondly, the main query aggregates the results from the "OrderSummary" CTE, grouping them by the customer's company name. It counts the number of orders and sums up the total sales amount for each customer.This approach not only streamlines the code but also improves its maintainability and understandability. By breaking down the problem into logical steps and using meaningful aliases for tables and columns, the query becomes self-explanatory even for someone not deeply familiar with the database schema.

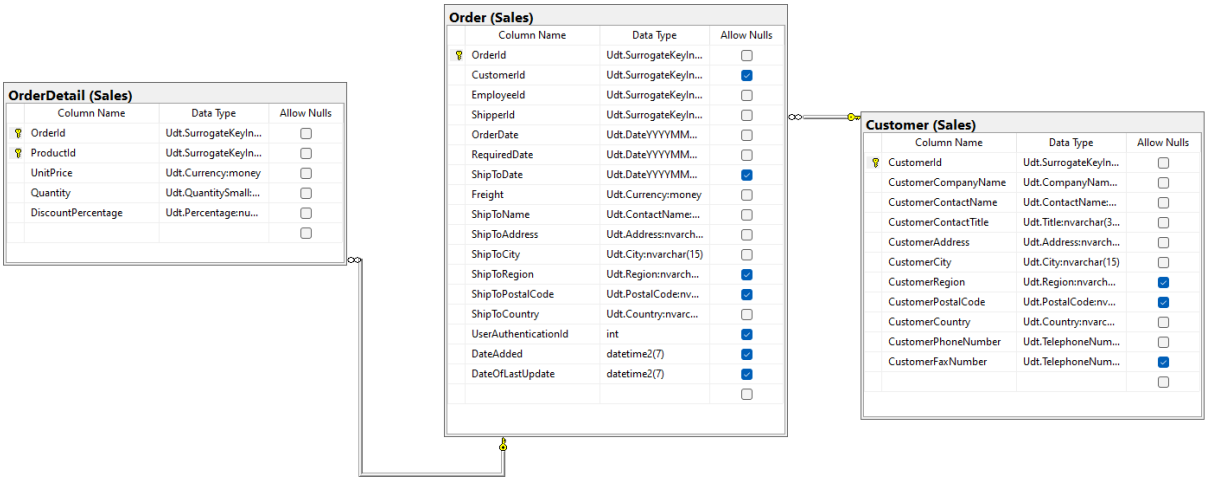
In [ ]:
```
USE Northwinds2022TSQLV7;

WITH OrderSummary AS (
    SELECT
        c.CustomerCompanyName,
        o.OrderId,
        o.OrderDate,
        SUM(od.UnitPrice * od.Quantity * (1 - od.DiscountPercentage)) AS TotalAmount
    FROM
        Sales.Customer c
    INNER JOIN
        Sales.[Order] o ON c.CustomerId = o.CustomerId
    INNER JOIN
        Sales.OrderDetail od ON o.OrderId = od.OrderId
    GROUP BY
        c.CustomerCompanyName, o.OrderId, o.OrderDate
)
SELECT
    CustomerCompanyName,
    COUNT(OrderId) AS TotalOrders,
    SUM(TotalAmount) AS TotalSales
FROM
    OrderSummary
GROUP BY
    CustomerCompanyName;
```

Table Diagram :



Column Standards of tables:



--17

Query 3:

Proposition: Analyzing Product Distribution by Category and Supplier Country

The main objective is to summarize the data by category and supplier country, presenting the total number of products and the count of unique suppliers for each category-country combination

Why is the top Problem?

It efficiently provides crucial insights into product distribution across categories and countries. By combining data from the Product, Category, and Supplier tables, it offers a comprehensive view of product-supplier relationships.

Firstly, it selects relevant fields like ProductName, CategoryName, SupplierCompanyName, and SupplierCountry, providing a clear understanding of the products, their categories, and their respective suppliers.

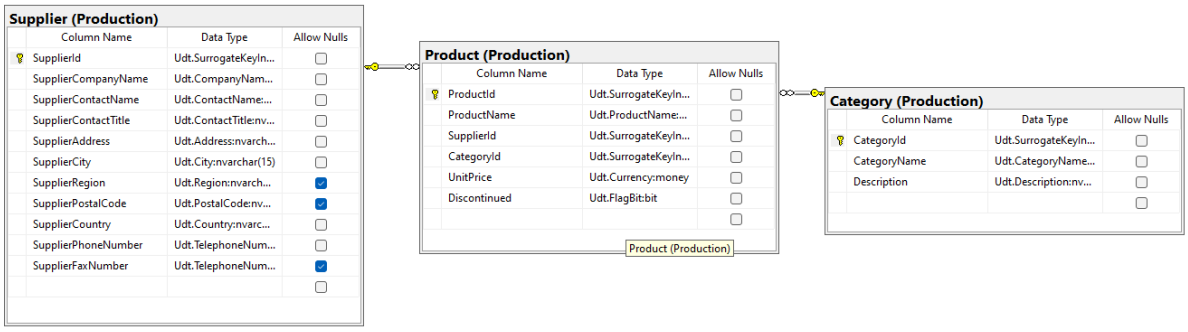
Next, it counts the total number of products and the number of unique suppliers within each category and country combination. This allows for a precise evaluation of product diversity and supplier distribution across different regions.

Lastly, the results are sorted by CategoryName and SupplierCountry, ensuring a structured presentation of the data for easy analysis and comparison.

In [ ]:
```
USE Northwinds2022TSQLV7;

WITH ProductSupplierDetails AS (
    SELECT
        p.ProductId,
        p.ProductName,
        c.CategoryName,
        s.SupplierCompanyName,
        s.SupplierCountry
    FROM
        Production.Product p
    INNER JOIN
        Production.Category c ON p.CategoryId = c.CategoryId
    INNER JOIN
        Production.Supplier s ON p.SupplierId = s.SupplierId
)
SELECT
    CategoryName,
    SupplierCountry,
    COUNT(ProductId) AS TotalProducts,
    COUNT(DISTINCT SupplierCompanyName) AS UniqueSuppliers
FROM
    ProductSupplierDetails
GROUP BY
    CategoryName, SupplierCountry
ORDER BY
    CategoryName, SupplierCountry;
```

Column Standards of Tables:



Top 3 Worst Problems

--20

Query 4:

Proposition: Retrieves the latest salary information for each employee along with their current and previous salary details and identifies whether there was an increase, decrease, or no change in salary.

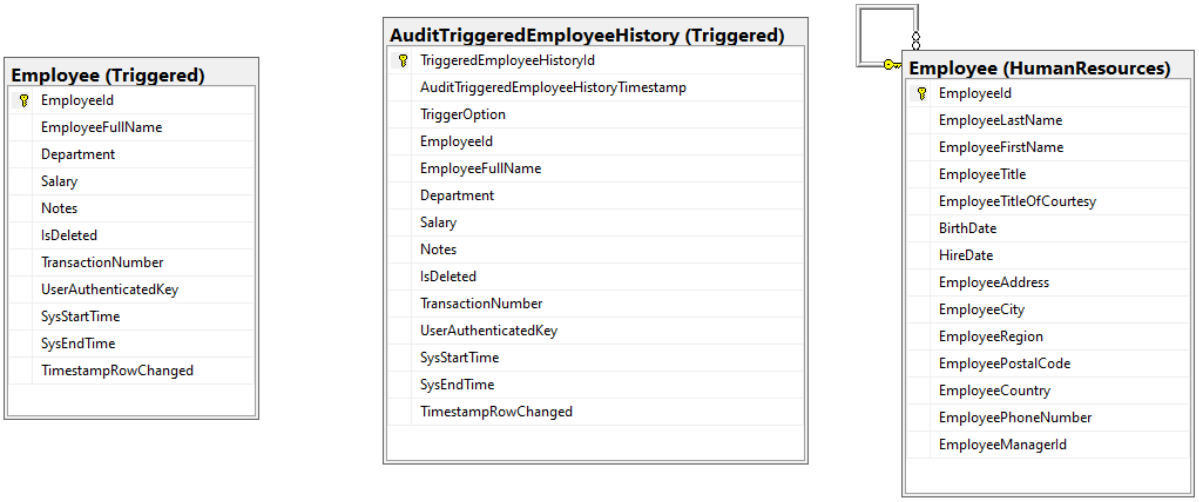
Why is it worst problem?

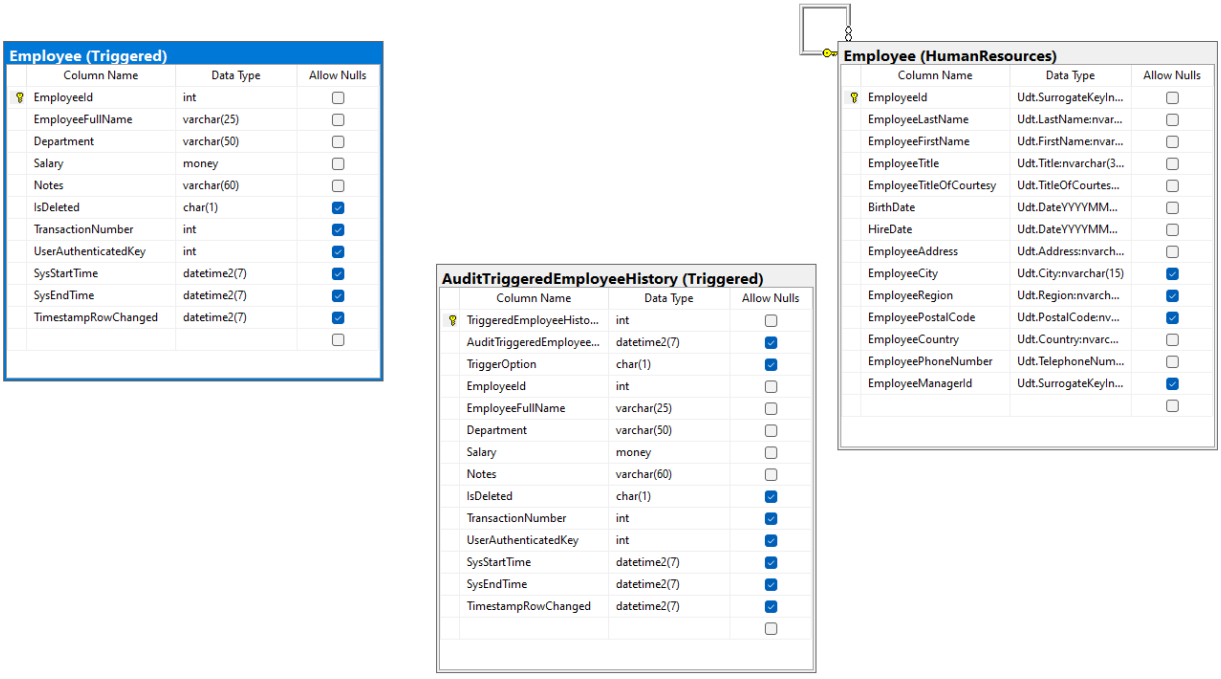
1. **Complexity**: The query involves multiple CTEs (Common Table Expressions) and joins, which can make it challenging to understand and maintain. If someone unfamiliar with the database structure or the purpose of the query needs to work on it, they might struggle to comprehend its logic.
2. **Performance**: With multiple joins and calculations, this query might suffer from performance issues, especially as the data volume grows. Poorly optimized queries can slow down the database and impact other applications relying on it.
3. **Maintenance**: Any changes to the database schema or requirements could potentially break this query. Since it's quite intricate, even small modifications might require significant effort to ensure its continued functionality.
4. **Clarity**: The query lacks sufficient comments or documentation to explain its purpose and logic. Without proper documentation, it becomes even more challenging for developers to understand and troubleshoot it.
5. **Data Integrity**: Depending on how the historical data is managed and updated, there's a risk of inconsistencies or inaccuracies in the results. Ensuring data integrity in such complex queries can be a significant challenge.

In [ ]:
```
USE Northwinds2022TSQLV7;

WITH LatestTriggeredEmployeeHistory AS (
    SELECT
        t1.EmployeeId,
```

```
        t1.EmployeeFullName,
        t1.Department,
        t1.Salary AS CurrentSalary,
        t1.SysEndTime AS CurrentSalaryEndTime,
        t2.Salary AS PreviousSalary,
        t2.SysEndTime AS PreviousSalaryEndTime,
        ROW_NUMBER() OVER (PARTITION BY t1.EmployeeId ORDER BY t1.SysEndTime DESC) AS Ro
    FROM
        Triggered.Employee AS t1
    LEFT JOIN
        Triggered.AuditTriggeredEmployeeHistory AS t2
    ON
        t1.EmployeeId = t2.EmployeeId
    WHERE
        t1.IsDeleted = 'N'
        AND t1.SysEndTime = '9999-12-31T23:59:59'
),
EmployeeSalaryChange AS (
    SELECT
        EmployeeId,
        EmployeeFullName,
        Department,
        CurrentSalary,
        CurrentSalaryEndTime,
        PreviousSalary,
        PreviousSalaryEndTime,
        CASE
            WHEN CurrentSalary > PreviousSalary THEN 'Increase'
            WHEN CurrentSalary < PreviousSalary THEN 'Decrease'
            ELSE 'No Change'
        END AS SalaryChangeType
    FROM
        LatestTriggeredEmployeeHistory
    WHERE
        RowNum = 1
)
SELECT
    e.EmployeeId,
    e.EmployeeLastName,
    e.EmployeeFirstName,
    e.EmployeeTitle,
    e.EmployeeTitleOfCourtesy,
    e.BirthDate,
    e.HireDate,
    e.EmployeeAddress,
    e.EmployeeCity,
    e.EmployeeRegion,
    e.EmployeePostalCode,
    e.EmployeeCountry,
    e.EmployeePhoneNumber,
    e.EmployeeManagerId,
    t.EmployeeFullName,
    t.Department,
    t.CurrentSalary,
    t.PreviousSalary,
    t.SalaryChangeType
FROM
    HumanResources.Employee AS e
JOIN
    EmployeeSalaryChange AS t
ON
    e.EmployeeId = t.EmployeeId;
```





--6

Query 5: Proposition: Retrieves the count of addresses in each state or province along with their respective state/province name, code, and country region code, combines the Person Address and Person StateProvince tables, order by the address count in descending order, providing insight into the distribution of addresses across different states or provinces.

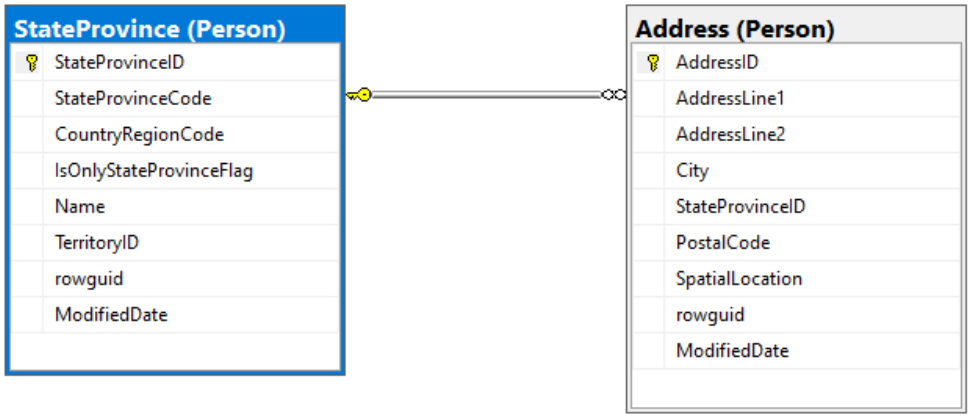
1. **Why is it worst problem?**

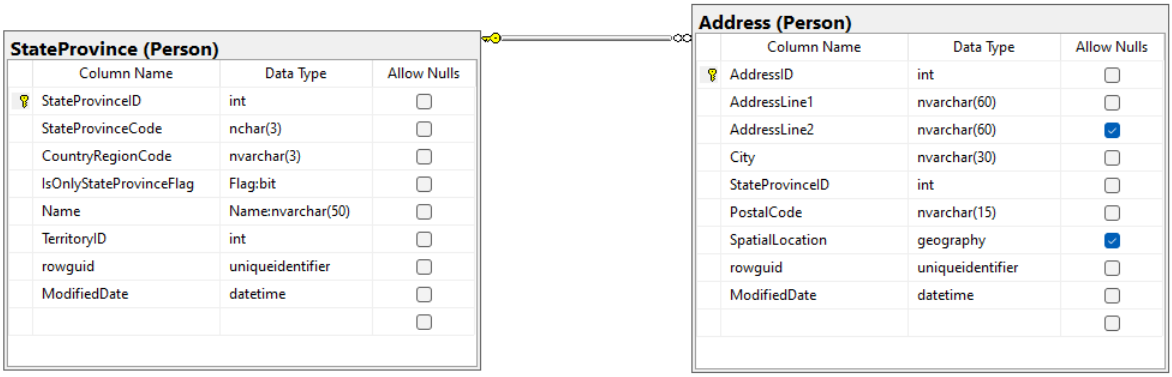
2. **Limited Scalability**: As the dataset grows, the query's performance might degrade significantly, making it unsuitable for handling larger volumes of data without optimizations.

3. **Inadequate Documentation**: As previously mentioned, the absence of documentation makes it difficult for developers to understand the query's purpose, assumptions, and potential limitations, leading to confusion and inefficiency during maintenance or troubleshooting.

In [ ]:
```
use AdventureWorks2017;

SELECT
    sp.Name AS StateProvinceName,
    sp.StateProvinceCode,
    sp.CountryRegionCode,
    COUNT(a.AddressID) AS AddressCount
FROM
    Person.Address a
INNER JOIN
    Person.StateProvince sp ON a.StateProvinceID = sp.StateProvinceID
GROUP BY
    sp.Name,
    sp.StateProvinceCode,
    sp.CountryRegionCode
ORDER BY
    AddressCount DESC;
```





--19

Query 6:

Proposition: Exploring Employee Performance and Customer Demographics

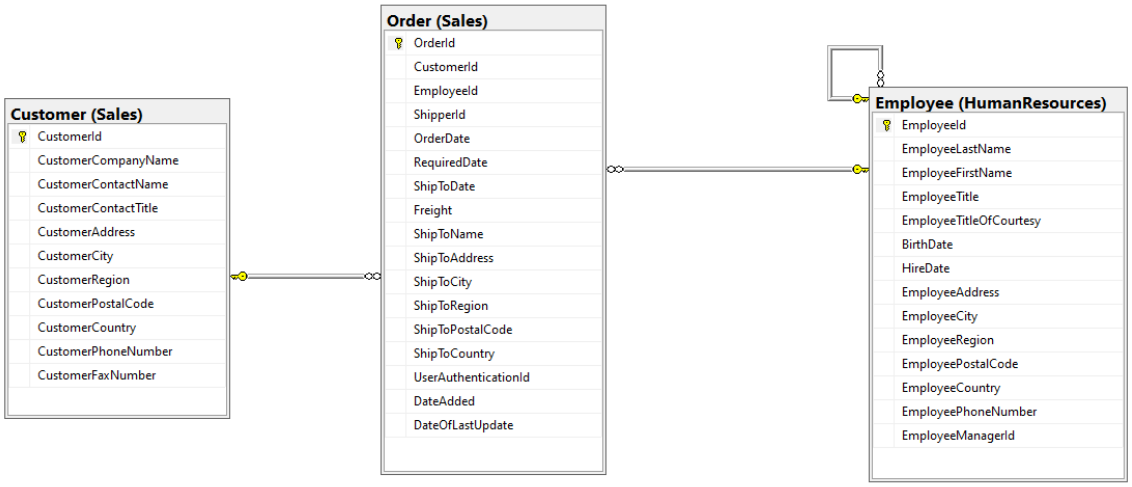
Why is it worst problem?

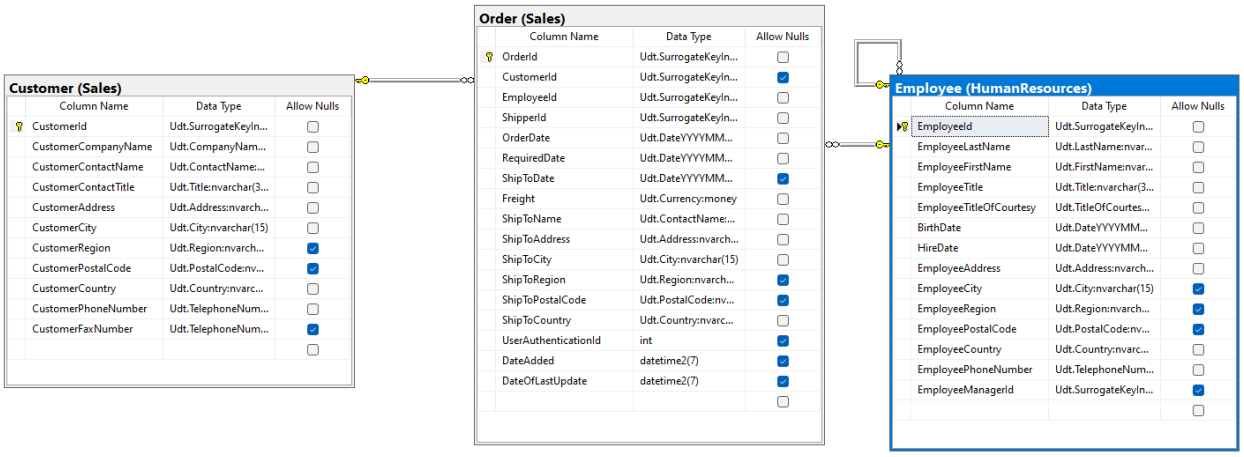
This SQL script is problematic because it's overly verbose, lacks comments for clarity, may have performance issues due to excessive JOINs and CTEs, features inconsistent naming conventions, and ends with a potentially confusing ordering choice.

In [ ]:
```
USE Northwinds2022TSQLV7;

WITH OrderDetails AS (
    SELECT
        o.OrderId,
        o.CustomerId,
        o.EmployeeId,
        o.OrderDate,
        o.Freight,
        c.CustomerCountry,
        e.EmployeeTitle,
        e.EmployeeCountry,
        ROW_NUMBER() OVER (PARTITION BY o.EmployeeId ORDER BY o.OrderDate DESC) AS Order
    FROM
        Sales.[Order] o
    INNER JOIN
        Sales.Customer c ON o.CustomerId = c.CustomerId
    INNER JOIN
        HumanResources.Employee e ON o.EmployeeId = e.EmployeeId
),
TopEmployees AS (
    SELECT
        EmployeeId,
        COUNT(OrderId) AS TotalOrders,
        SUM(Freight) AS TotalFreight,
        MAX(OrderRank) AS HighestOrderRank
    FROM
        OrderDetails
```

```
    GROUP BY
        EmployeeId
),
CustomerDemographics AS (
    SELECT
        CustomerCountry,
        COUNT(CustomerId) AS TotalCustomers
    FROM
        Sales.Customer
    GROUP BY
        CustomerCountry
)
SELECT
    e.EmployeeId,
    e.EmployeeLastName,
    e.EmployeeFirstName,
    e.EmployeeTitle,
    e.EmployeeCountry AS EmployeeCountryOfOrigin,
    te.TotalOrders,
    te.TotalFreight,
    cd.CustomerCountry,
    cd.TotalCustomers
FROM
    HumanResources.Employee e
LEFT JOIN
    TopEmployees te ON e.EmployeeId = te.EmployeeId
LEFT JOIN
    CustomerDemographics cd ON e.EmployeeCountry = cd.CustomerCountry
ORDER BY
    te.TotalOrders DESC;
```





Remaining 14 Problems

--2

Query 7:

Proposition: Retrieving the total number of orders and total freight cost for each shipper. Summarized result with shipper information, including company name and phone number.

In [ ]:

```
use Northwinds2022TSQLV7;

WITH OrderSummary AS (
    SELECT
        o.ShipperId,
        COUNT(o.OrderId) AS TotalOrders,
        SUM(o.Freight) AS TotalFreight
    FROM
        Sales.[Order] o
    GROUP BY
        o.ShipperId
),
ShipperInfo AS (
    SELECT
        s.ShipperId,
        s.ShipperCompanyName,
        s.PhoneNumber
    FROM
        Sales.Shipper s
)
SELECT
    s.ShipperId,
    s.ShipperCompanyName,
    s.PhoneNumber,
    os.TotalOrders,
    os.TotalFreight
FROM
    ShipperInfo s
JOIN
    OrderSummary os ON s.ShipperId = os.ShipperId;
```

--4

Query 8:

Proposition: Retrieves order details along with the total revenue generated by each order.

In [ ]:
```
use Northwinds2022TSQLV7;

WITH OrderRevenue AS (
    SELECT
        o.OrderId,
        SUM(od.UnitPrice * od.Quantity * (1 - od.DiscountPercentage)) AS TotalRevenue
    FROM
        Sales.[Order] o
    INNER JOIN
        Sales.OrderDetail od ON o.OrderId = od.OrderId
    GROUP BY
        o.OrderId
)
SELECT
    o.OrderId,
    od.ProductId,
    od.UnitPrice,
    od.Quantity,
    od.DiscountPercentage,
    OrderRevenue.TotalRevenue
FROM
    Sales.[Order] o
INNER JOIN
    Sales.OrderDetail od ON o.OrderId = od.OrderId
INNER JOIN
    OrderRevenue  ON o.OrderId = OrderRevenue.OrderId;
```

--5

Query 9:

Proposition: This query retrieves the latest pay rate change date and the corresponding pay rate for each job candidate.

In [ ]:
```
USE AdventureWorks2017;

SELECT
    jc.JobCandidateID,
    jc.BusinessEntityID,
    jc.Resume,
    eph.RateChangeDate,
    eph.Rate
FROM
    HumanResources.JobCandidate jc
LEFT JOIN (
    SELECT
        BusinessEntityID,
        MAX(RateChangeDate) AS LatestChangeDate
    FROM
        HumanResources.EmployeePayHistory
    GROUP BY
        BusinessEntityID
) AS lprc ON jc.BusinessEntityID = lprc.BusinessEntityID
LEFT JOIN
```

```
    HumanResources.EmployeePayHistory eph ON lprc.BusinessEntityID = eph.BusinessEntityI
                                          AND lprc.LatestChangeDate = eph.RateChangeD
```

--10

Query 10:

Proposition (Complex) : Analyze the sales performance of each customer:

This SQL query combines data from the Fact.Sale, Dimension.Customer, and Fact.Order tables to analyze the sales performance of each customer. Specifically analyzes the sales performance for "Tailspin Toys (Head Office)" and "Tailspin Toys (Peeples Valley, AZ)" customers. It calculates their total sales value, the number of sales, and the total number of orders.

In [ ]:
```
USE WideWorldImportersDW;

WITH CustomerSales AS (
    SELECT
        C.[Customer Key],
        C.Customer,
        SUM(S.[Total Including Tax]) AS TotalSalesValue,
        COUNT(S.[Sale Key]) AS TotalSalesCount
    FROM
        Fact.Sale S
    INNER JOIN
        Dimension.Customer C ON S.[Customer Key] = C.[Customer Key]
    WHERE
        C.Customer IN ('Tailspin Toys (Head Office)', 'Tailspin Toys (Peeples Valley, AZ
    GROUP BY
        C.[Customer Key], C.Customer
),
CustomerOrders AS (
    SELECT
        C.[Customer Key],
        COUNT(O.[Order Key]) AS TotalOrders
    FROM
        Fact.[Order] O
    INNER JOIN
        Dimension.Customer C ON O.[Customer Key] = C.[Customer Key]
    WHERE
        C.Customer IN ('Tailspin Toys (Head Office)', 'Tailspin Toys (Peeples Valley, AZ
    GROUP BY
        C.[Customer Key]
)
SELECT
    CS.[Customer Key],
    CS.Customer,
    CS.TotalSalesValue,
    CS.TotalSalesCount,
    CO.TotalOrders
FROM
    CustomerSales CS
LEFT JOIN
    CustomerOrders CO ON CS.[Customer Key] = CO.[Customer Key]
ORDER BY
    CS.TotalSalesValue DESC;
```

--7

Query 11:

7) Proposition: This SQL query retrieves unique business entities of person associated with addresses in the city of Monroe.

Proposition: This SQL query retrieves unique business entities of person associated with addresses in the city of Monroe.

In [ ]:
```
use AdventureWorks2017;
SELECT DISTINCT
    bea.BusinessEntityID,
    a.AddressLine1,
    a.AddressLine2,
    a.City,
    a.StateProvinceID,
    a.PostalCode
FROM
    Person.BusinessEntityAddress bea
INNER JOIN
    Person.Address a ON bea.AddressID = a.AddressID
WHERE
    a.City = 'Monroe';
```

--8

Query 12:

Proposition: Retrieve business id who has vista credit card which expires in 2008 and show their credit card id and number.

In [ ]:
```
Use AdventureWorks2017;

SELECT
    pc.BusinessEntityID,
    pc.CreditCardID,
    c.CardNumber,
    c.CardType
FROM
    Sales.PersonCreditCard pc
INNER JOIN
    Sales.CreditCard c ON pc.CreditCardID = c.CreditCardID
WHERE
    c.CardType = 'Vista'
    AND c.ExpYear = 2008;
```

--9

Query 13:

Proposition : Analyze sales data by calculating the total value of orders and the total quantity of items ordered for each customer and identify the top-selling items and their contribution to overall revenue.

In [ ]:
```
USE WideWorldImporters;
```

```
WITH TopSellingItems AS (
    SELECT
        OL.StockItemID,
        SUM(OL.Quantity) AS TotalSoldQuantity,
        SUM(OL.Quantity * OL.UnitPrice) AS TotalRevenue,
        ROW_NUMBER() OVER (ORDER BY SUM(OL.Quantity * OL.UnitPrice) DESC) AS RowNum
    FROM
        Sales.OrderLines OL
    GROUP BY
        OL.StockItemID
)
SELECT
    StockItemID,
    TotalSoldQuantity,
    TotalRevenue
FROM
    TopSellingItems
WHERE
    RowNum <= 5;
```

--11

Query 14:

Proposition: Analyze the movement data for customers belonging to the 'Tailspin Toys' buying group. Specifically, identify the top 3 and bottom 3 customers based on their total movements.

In [ ]:
```
USE WideWorldImportersDW;

WITH CustomerMovements AS (
    SELECT
        C.Customer,
        C.[Customer Key],
        SUM(M.Quantity) AS TotalMovements
    FROM
        Dimension.Customer C
    JOIN
        Fact.Movement M ON C.[Customer Key] = M.[Customer Key]
    WHERE
        C.[Buying Group] = 'Tailspin Toys'
    GROUP BY
        C.Customer, C.[Customer Key]
),
TopCustomers AS (
    SELECT
        CM.Customer,
        CM.[Customer Key],
        CM.TotalMovements,
        ROW_NUMBER() OVER (ORDER BY CM.TotalMovements DESC) AS MovementRank
    FROM
        CustomerMovements CM
),
BottomCustomers AS (
    SELECT
        CM.Customer,
        CM.[Customer Key],
        CM.TotalMovements,
        ROW_NUMBER() OVER (ORDER BY CM.TotalMovements ASC) AS MovementRank
    FROM
        CustomerMovements CM
)
```

```
SELECT
    TC.Customer,
    TC.[Customer Key],
    TC.TotalMovements AS TotalMovementsForTopCustomer,
    BC.TotalMovements AS TotalMovementsForBottomCustomer
FROM
    TopCustomers TC
FULL JOIN
    BottomCustomers BC ON TC.MovementRank = BC.MovementRank
WHERE
    TC.MovementRank <= 3 OR BC.MovementRank <= 3
ORDER BY
    TC.MovementRank;
```

--12

Query 15

Proposition: Retrieve Essential Customer Details for Newcastle, New South Wales, Australia

In [ ]:
```
USE AdventureWorksDW2017;

SELECT
    dc.CustomerKey,
    dc.FirstName,
    dc.LastName,
    dc.EmailAddress,
    dc.YearlyIncome,
    dc.TotalChildren,
    dc.NumberChildrenAtHome,
    dc.EnglishEducation,
    dc.EnglishOccupation,
    dc.HouseOwnerFlag,
    dc.NumberCarsOwned,
    dc.AddressLine1,
    dc.Phone
FROM
    dbo.DimCustomer AS dc
JOIN
    dbo.DimGeography AS dg ON dc.GeographyKey = dg.GeographyKey
WHERE
    dg.City = 'Newcastle'
    AND dg.StateProvinceName = 'New South Wales'
    AND dg.CountryRegionCode = 'AU';
```

--13

Query 16:

Peoposition: Retrieve summarized product information, including safety stock levels, categorized by English product category names.

In [ ]:
```
USE AdventureWorksDW2017;

WITH ProductDetails AS (
    SELECT
        p.ProductKey,
        p.ProductAlternateKey,
        p.SafetyStockLevel,
        ps.EnglishProductSubcategoryName,
```

```
        pc.EnglishProductCategoryName
    FROM
        dbo.DimProduct p
    INNER JOIN
        dbo.DimProductSubcategory ps ON p.ProductSubcategoryKey = ps.ProductSubcategoryK
    INNER JOIN
        dbo.DimProductCategory pc ON ps.ProductCategoryKey = pc.ProductCategoryKey
),
CalcSafetyStock AS (
    SELECT
        EnglishProductCategoryName,
        SUM(SafetyStockLevel) AS TotalSafetyStock
    FROM
        ProductDetails
    GROUP BY
        EnglishProductCategoryName
)
SELECT
    EnglishProductCategoryName,
    TotalSafetyStock
FROM
    CalcSafetyStock
ORDER BY
    EnglishProductCategoryName;
```

--14

Query 17

Proposition: Analysis of Average End-of-Day Currency Rates Across Organizations

In [ ]:
```
USE AdventureWorksDW2017;

WITH AvgEndOfDayRates AS (
    SELECT
        c.CurrencyName,
        o.OrganizationName,
        AVG(f.EndOfDayRate) AS AvgEndOfDayRate
    FROM
        dbo.FactCurrencyRate f
    INNER JOIN
        dbo.DimCurrency c ON f.CurrencyKey = c.CurrencyKey
    INNER JOIN
        dbo.DimOrganization o ON c.CurrencyKey = o.CurrencyKey
    GROUP BY
        c.CurrencyName, o.OrganizationName
)
SELECT
    CurrencyName,
    AVG(AvgEndOfDayRate) AS OverallAvgEndOfDayRate
FROM
    AvgEndOfDayRates
GROUP BY
    CurrencyName;
```

--15

Query 18:

Proposition: Analysis of Currency Rate Fluctuations Over Time

This query calculates the daily rate fluctuations for each currency and identifies significant fluctuations based on a predefined threshold. It then aggregates the significant fluctuations by year and month, providing insights into the average fluctuation for each currency over time.

In [ ]:
```
USE AdventureWorksDW2017;

WITH CurrencyFluctuations AS (
    SELECT
        DC.CurrencyName,
        FCR.Date,
        FCR.EndOfDayRate - LAG(FCR.EndOfDayRate) OVER (PARTITION BY FCR.CurrencyKey ORDE
    FROM
        dbo.FactCurrencyRate FCR
    JOIN
        dbo.DimCurrency DC ON FCR.CurrencyKey = DC.CurrencyKey
),
SignificantFluctuations AS (
    SELECT
        CurrencyName,
        Date,
        RateFluctuation
    FROM
        CurrencyFluctuations
    WHERE
        ABS(RateFluctuation) > 0.05 /* Adjust threshold as needed */
)
SELECT
    CurrencyName,
    DATEPART(YEAR, Date) AS Year,
    DATEPART(MONTH, Date) AS Month,
    AVG(RateFluctuation) AS AverageFluctuation
FROM
    SignificantFluctuations
GROUP BY
    CurrencyName,
    DATEPART(YEAR, Date),
    DATEPART(MONTH, Date)
ORDER BY
    CurrencyName,
    Year,
    Month;
```

--16

Query 19:

Proposition: perform a comparative analysis of currency utilization across different organizations.

This query calculates the percentage of total transactions conducted in the primary currency for each organization. It identifies the primary currency for each organization and calculates the percentage of total transactions conducted in that currency. Finally, it filters the results to include only the primary currency for each organization.

In [ ]:
```
USE AdventureWorksDW2017;

WITH CurrencyUtilization AS (
```

```
    SELECT
        O.OrganizationName,
        DC.CurrencyName AS PrimaryCurrency,
        COUNT(*) AS TotalTransactions
    FROM
        dbo.DimOrganization O
    JOIN
        dbo.DimCurrency DC ON O.CurrencyKey = DC.CurrencyKey
    GROUP BY
        O.OrganizationName,
        DC.CurrencyName
),
PrimaryCurrencyUtilization AS (
    SELECT
        OrganizationName,
        PrimaryCurrency,
        TotalTransactions,
        ROW_NUMBER() OVER (PARTITION BY OrganizationName ORDER BY TotalTransactions DESC
    FROM
        CurrencyUtilization
)
SELECT
    OrganizationName,
    PrimaryCurrency,
    TotalTransactions,
    ROUND((TotalTransactions / SUM(TotalTransactions) OVER (PARTITION BY OrganizationNam
FROM
    PrimaryCurrencyUtilization
WHERE
    Rank = 1; /* Filter for primary currency */
```

--18

Query 20:

Proposition: Analyzing Top Suppliers by Total Revenue and Total Products Supplied, identify the top-performing suppliers based on total revenue

In [ ]:

```
USE Northwinds2022TSQLV7;

WITH SupplierDetails AS (
    SELECT
        s.SupplierId,
        s.SupplierCompanyName,
        s.SupplierCountry,
        COUNT(p.ProductId) AS TotalProductsSupplied,
        SUM(p.UnitPrice) AS TotalRevenue
    FROM
        Production.Supplier s
    LEFT JOIN
        Production.Product p ON s.SupplierId = p.SupplierId
    GROUP BY
        s.SupplierId,
        s.SupplierCompanyName,
        s.SupplierCountry
),
TopSuppliers AS (
    SELECT
        *,
        ROW_NUMBER() OVER (ORDER BY TotalRevenue DESC) AS SupplierRank
    FROM
        SupplierDetails
)
```

```sql
SELECT top(5)
    SupplierRank,
    SupplierCompanyName,
    SupplierCountry,
    TotalProductsSupplied,
    TotalRevenue
FROM
    TopSuppliers
```