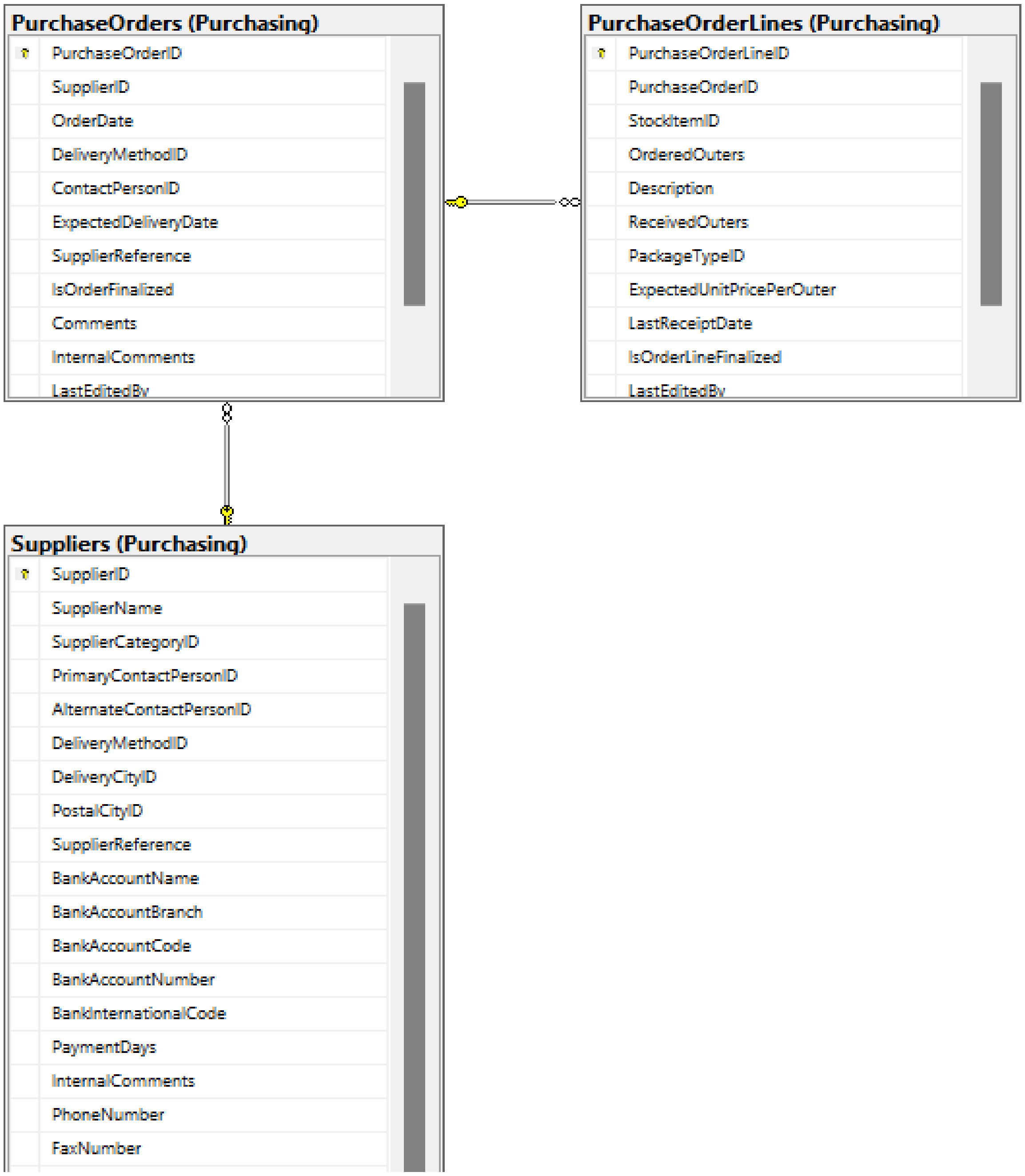
PROJECT ONE

- This project aims to utilize database diagrams as key navigation and analysis tools, develop diagram views to separate subsystems for specific information queries, apply diagrammatic subject areas to address real-world business problems, document necessary data for accurate query resolution. Each of the five group member creates 20 queries across six selected databases. Use organized diagram views to facilitate the exploration and solving of complex issues.

-

TOP QUERY (1)

USE: WideWorldimporters Database



PROPOSITION: Create or update a view named Purchasing.SupplierPerformanceStats to store summarized performance metrics for suppliers based on their historical

Columns from their respective tables in the select clause:

Table Name	Column Name
Purchasing.Suppliers,,	SupplierID, SupplierName
Purchasing.PurchaseOrders	SupplierID (used for JOIN with Purchasing.Suppliers) OrderDate PurchaseOrderID
Purchasing.PurchaseOrderLines	PurchaseOrderID (used for JOIN with Purchasing.PurchaseOrders) ExpectedUnitPricePerOuter OrderedOuters

Order by :

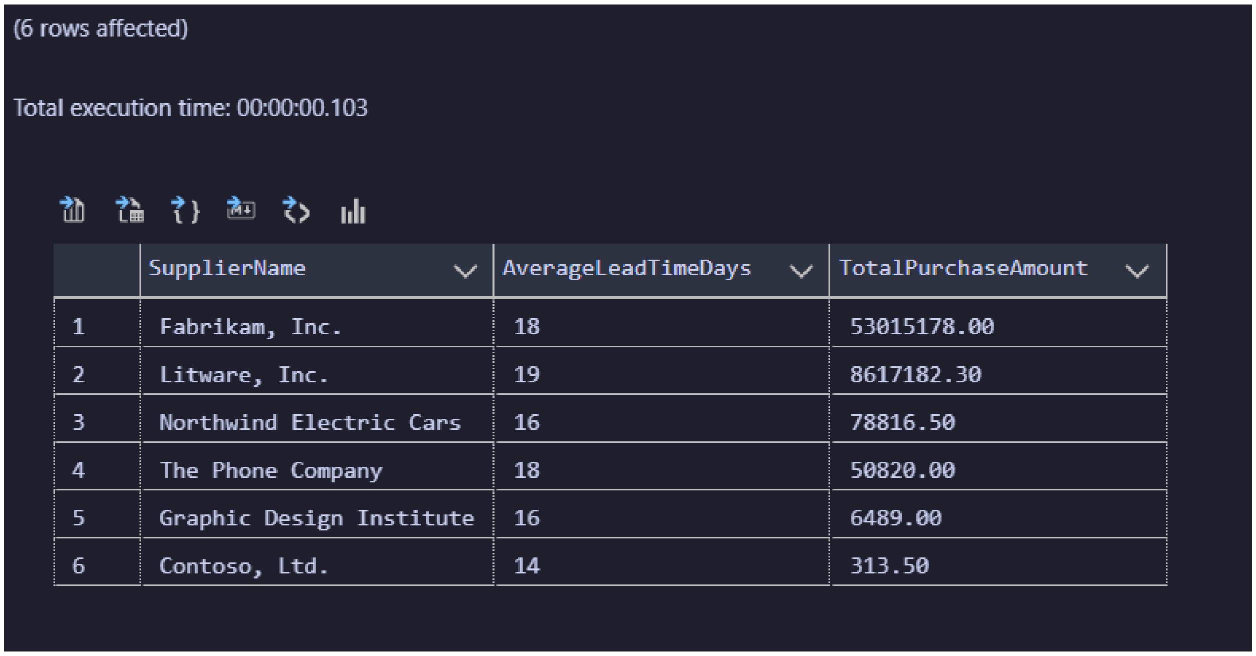
Table Name	Column Name	Sort Order
Purchasing.Suppliers	TotalPurchaseAmount	Descending

Problem solving Query:

```
USE WideWorldimporters; -- MEDIUM QUERY

GO


WITH SupplierPerformance AS (

    SELECT

        s.SupplierID,

        s.SupplierName,

        AVG(DATEDIFF(day, po.OrderDate, po.ExpectedDeliveryDate)) AS AverageLeadTimeDays,

        SUM(pol.ExpectedUnitPricePerOuter * pol.OrderedOuters) AS TotalPurchaseAmount

    FROM

        Purchasing.Suppliers s

    INNER JOIN Purchasing.PurchaseOrders po ON s.SupplierID = po.SupplierID
```

```sql
    INNER JOIN Purchasing.PurchaseOrderLines pol ON po.PurchaseOrderID =
pol.PurchaseOrderID

    WHERE

        po.OrderDate BETWEEN '2013-01-01' AND '2013-12-31'

    GROUP BY

        s.SupplierID, s.SupplierName

)

SELECT

    SupplierName,

    AverageLeadTimeDays,

    TotalPurchaseAmount

FROM

    SupplierPerformance

ORDER BY

    TotalPurchaseAmount DESC

--FOR JSON PATH, ROOT('SupplierPerformance');
```

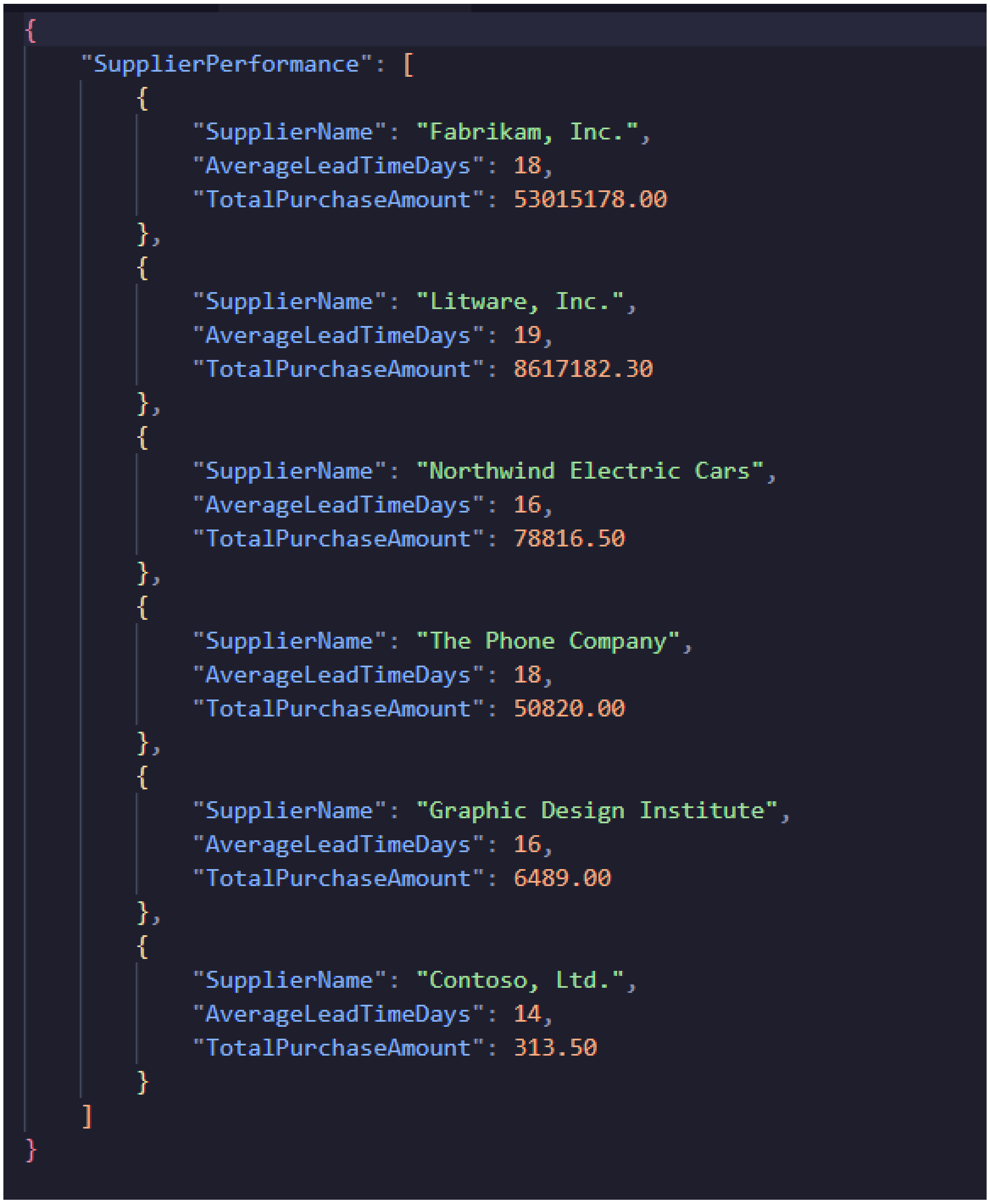
Sample Relational Output with total number of rows returned:



```
(6 rows affected)

Total execution time: 00:00:00.103
```

	SupplierName	AverageLeadTimeDays	TotalPurchaseAmount
1	Fabrikam, Inc.	18	53015178.00
2	Litware, Inc.	19	8617182.30
3	Northwind Electric Cars	16	78816.50
4	The Phone Company	18	50820.00
5	Graphic Design Institute	16	6489.00
6	Contoso, Ltd.	14	313.50

Sample JSON Output with total number of rows returned:

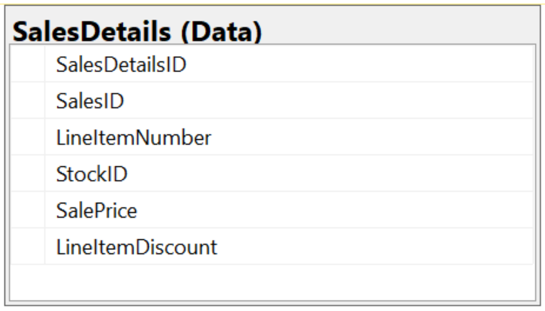


```
{
    "SupplierPerformance": [
        {
            "SupplierName": "Fabrikam, Inc.",
            "AverageLeadTimeDays": 18,
            "TotalPurchaseAmount": 53015178.00
        },
        {
            "SupplierName": "Litware, Inc.",
            "AverageLeadTimeDays": 19,
            "TotalPurchaseAmount": 8617182.30
        },
        {
            "SupplierName": "Northwind Electric Cars",
            "AverageLeadTimeDays": 16,
            "TotalPurchaseAmount": 78816.50
        },
        {
            "SupplierName": "The Phone Company",
            "AverageLeadTimeDays": 18,
            "TotalPurchaseAmount": 50820.00
        },
        {
            "SupplierName": "Graphic Design Institute",
            "AverageLeadTimeDays": 16,
            "TotalPurchaseAmount": 6489.00
        },
        {
            "SupplierName": "Contoso, Ltd.",
            "AverageLeadTimeDays": 14,
            "TotalPurchaseAmount": 313.50
        }
    ]
}
```

TOP QUERY (2)

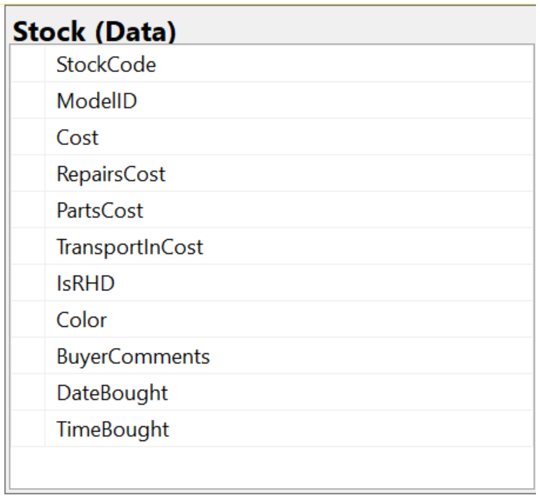
USE PrestsigeCars :-

NOTE DATABASE SCRIPT DOES NOT DEFINE FOREIGN KEY AND PRIMARY KEY RELATIONSHIP



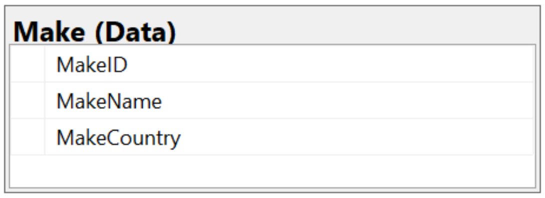
SalesDetails (Data)

- SalesDetailsID
- SalesID
- LineItemNumber
- StockID
- SalePrice
- LineItemDiscount



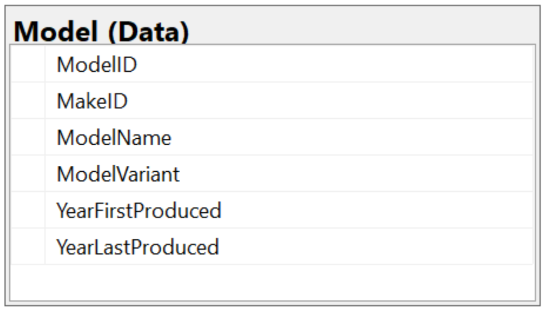
Stock (Data)

- StockCode
- ModelID
- Cost
- RepairsCost
- PartsCost
- TransportInCost
- IsRHD
- Color
- BuyerComments
- DateBought
- TimeBought



Make (Data)

- MakeID
- MakeName
- MakeCountry



Model (Data)

- ModelID
- MakeID
- ModelName
- ModelVariant
- YearFirstProduced
- YearLastProduced

Proposition: Develop or update a function named dbo.CalculateTotalCost within the PrestigeCars database. This function computes the comprehensive cost for a vehicle, integrating the initial cost with the expenses incurred from repairs, parts, and transport.

Columns from their respective tables in the select clause:

Table Name	Column Name
Data.Stock	Cost , StockCode, ModelID
Data.Model	ModelID, ModelName, MakeID
Data.Make	MakeID, MakeName
Data.SalesDetails	SalePrice, StockID, SalesID

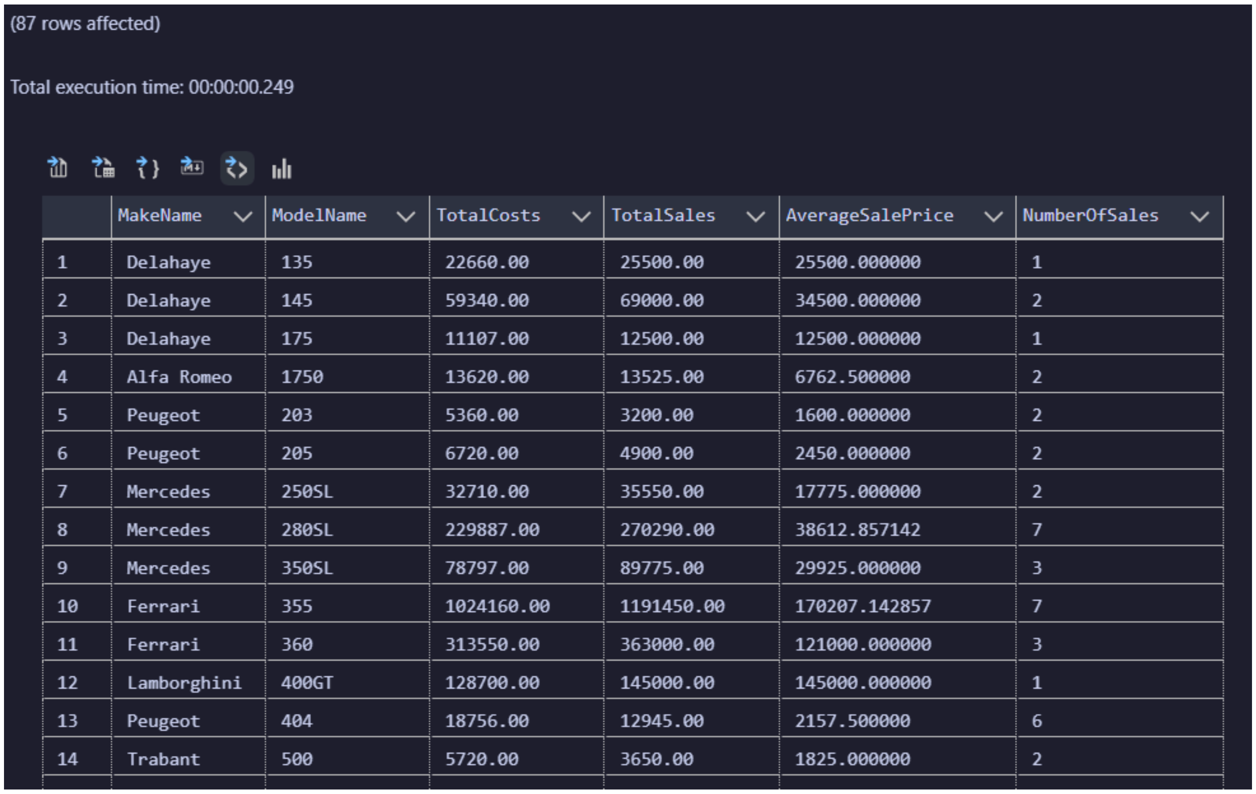
Order by :

Table Name	Column Name	Sort Order
SalesSummary CTE	TotalPurchaseAmount	Descending

Problem solving Query:

```
Use PrestigeCars -- COMPLEX QUERY
GO
CREATE OR ALTER FUNCTION dbo.CalculateTotalCost(
    @Cost money,
    @RepairsCost money,
    @PartsCost money,
    @TransportInCost money
)
RETURNS money
AS
BEGIN
    RETURN @Cost + @RepairsCost + @PartsCost + @TransportInCost;
END;
GO

WITH CarSalesAnalysis AS (
    SELECT
        MK.MakeName,
        MD.ModelName,
        dbo.CalculateTotalCost(ST.Cost, ST.RepairsCost, ST.PartsCost, ST.TransportInCost)
AS TotalCost,
        SD.SalePrice
    FROM Data.Stock ST
    INNER JOIN Data.Model MD ON ST.ModelID = MD.ModelID
    INNER JOIN Data.Make MK ON MD.MakeID = MK.MakeID
    INNER JOIN Data.SalesDetails SD ON ST.StockCode = SD.StockID
),
SalesSummary AS (
    SELECT
        MakeName,
        ModelName,
        SUM(TotalCost) AS TotalCosts,
        SUM(SalePrice) AS TotalSales,
        AVG(SalePrice) AS AverageSalePrice,
        COUNT(*) AS NumberOfSales
    FROM CarSalesAnalysis
    GROUP BY MakeName, ModelName
)
SELECT MakeName, ModelName, TotalCosts, TotalSales, AverageSalePrice, NumberOfSales
FROM SalesSummary
--FOR JSON PATH, ROOT('SalesSummary');
```

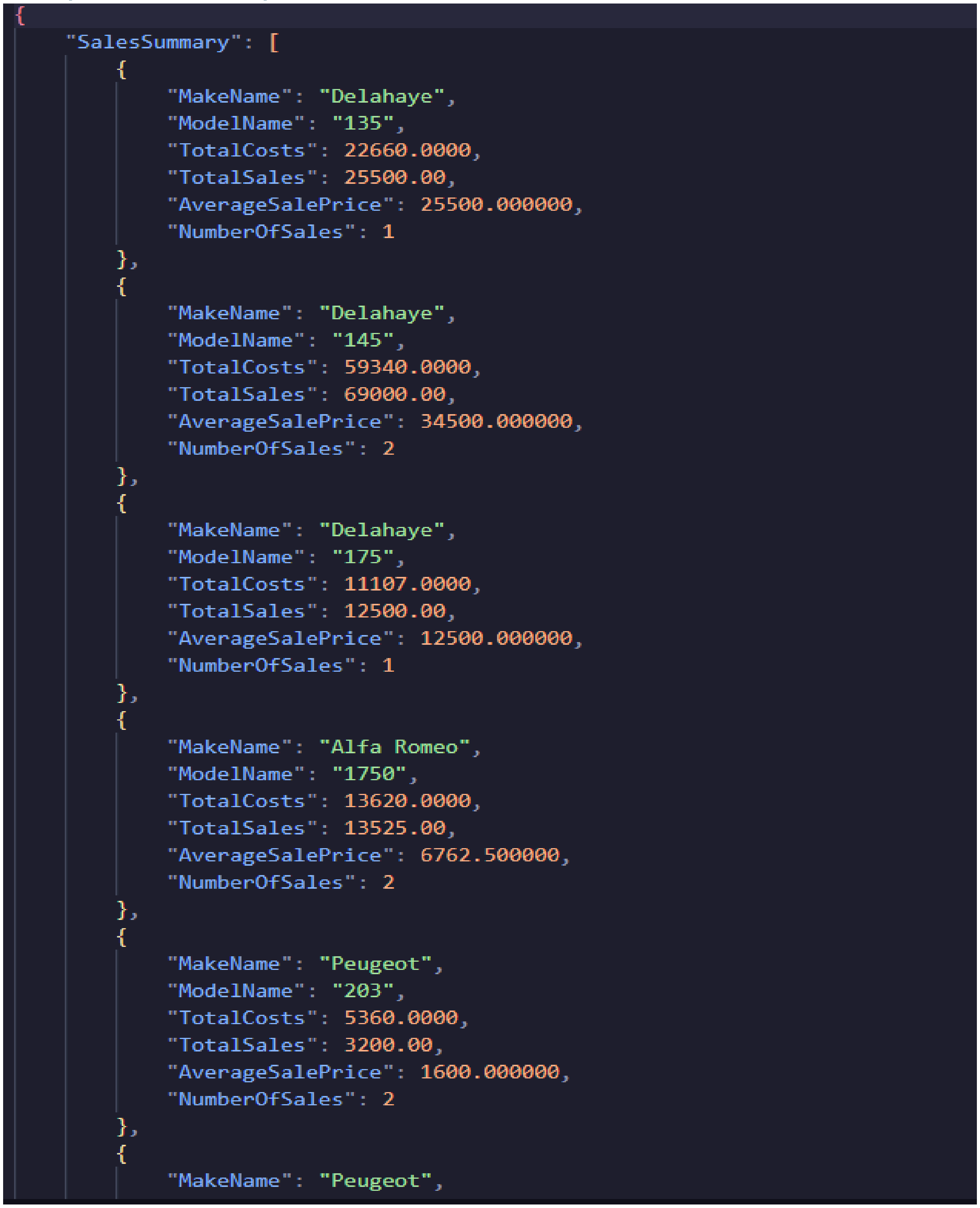
Sample Relational Output with total number of rows returned:



```
(87 rows affected)

Total execution time: 00:00:00.249
```

	MakeName	ModelName	TotalCosts	TotalSales	AverageSalePrice	NumberOfSales
1	Delahaye	135	22660.00	25500.00	25500.000000	1
2	Delahaye	145	59340.00	69000.00	34500.000000	2
3	Delahaye	175	11107.00	12500.00	12500.000000	1
4	Alfa Romeo	1750	13620.00	13525.00	6762.500000	2
5	Peugeot	203	5360.00	3200.00	1600.000000	2
6	Peugeot	205	6720.00	4900.00	2450.000000	2
7	Mercedes	250SL	32710.00	35550.00	17775.000000	2
8	Mercedes	280SL	229887.00	270290.00	38612.857142	7
9	Mercedes	350SL	78797.00	89775.00	29925.000000	3
10	Ferrari	355	1024160.00	1191450.00	170207.142857	7
11	Ferrari	360	313550.00	363000.00	121000.000000	3
12	Lamborghini	400GT	128700.00	145000.00	145000.000000	1
13	Peugeot	404	18756.00	12945.00	2157.500000	6
14	Trabant	500	5720.00	3650.00	1825.000000	2

Sample JSON Output with total number of rows returned:

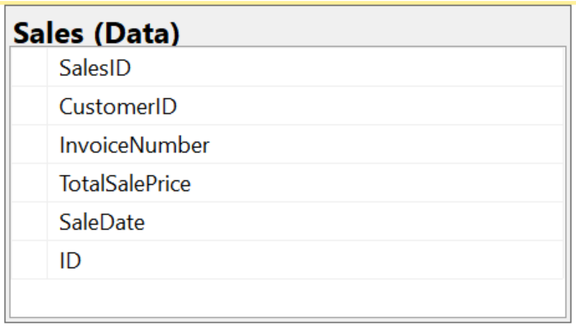


```
{
    "SalesSummary": [
        {
            "MakeName": "Delahaye",
            "ModelName": "135",
            "TotalCosts": 22660.0000,
            "TotalSales": 25500.00,
            "AverageSalePrice": 25500.000000,
            "NumberOfSales": 1
        },
        {
            "MakeName": "Delahaye",
            "ModelName": "145",
            "TotalCosts": 59340.0000,
            "TotalSales": 69000.00,
            "AverageSalePrice": 34500.000000,
            "NumberOfSales": 2
        },
        {
            "MakeName": "Delahaye",
            "ModelName": "175",
            "TotalCosts": 11107.0000,
            "TotalSales": 12500.00,
            "AverageSalePrice": 12500.000000,
            "NumberOfSales": 1
        },
        {
            "MakeName": "Alfa Romeo",
            "ModelName": "1750",
            "TotalCosts": 13620.0000,
            "TotalSales": 13525.00,
            "AverageSalePrice": 6762.500000,
            "NumberOfSales": 2
        },
        {
            "MakeName": "Peugeot",
            "ModelName": "203",
            "TotalCosts": 5360.0000,
            "TotalSales": 3200.00,
            "AverageSalePrice": 1600.000000,
            "NumberOfSales": 2
        },
        {
            "MakeName": "Peugeot",
```

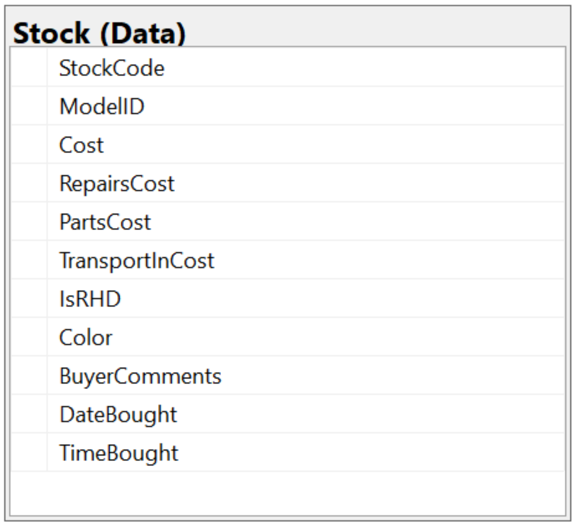
TOP QUERY (3):

Use PrestigeCars:

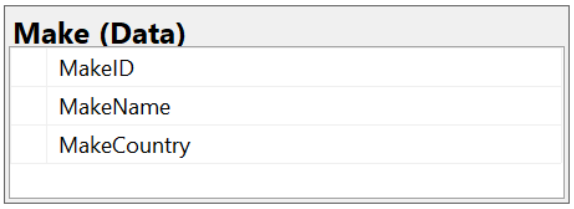
NOTE DATABASE SCRIPT DOES NOT DEFINE FOREIGN KEY AND PRIMARY KEY RELATIONSHIP



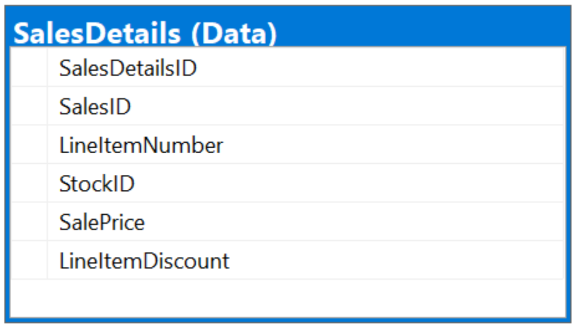
Sales (Data)
- SalesID
- CustomerID
- InvoiceNumber
- TotalSalePrice
- SaleDate
- ID



Stock (Data)
- StockCode
- ModelID
- Cost
- RepairsCost
- PartsCost
- TransportInCost
- IsRHD
- Color
- BuyerComments
- DateBought
- TimeBought



Make (Data)
- MakeID
- MakeName
- MakeCountry



SalesDetails (Data)
- SalesDetailsID
- SalesID
- LineItemNumber
- StockID
- SalePrice
- LineItemDiscount

Proposition: Create or revise a function named dbo.CalculateTotalCost in the PrestigeCars database. This function determines the total cost for a vehicle by incorporating the base sale price and any additional charges.

Columns from their respective tables in the select clause:

Table Name	Column Name
Data.Sales (SA)	SalesID
Data.SalesDetails (SD)	SalePrice, SalesID, StockID
Data.Stock (ST)	StockCode, ModelID
Data.Make (MK)	MakeID

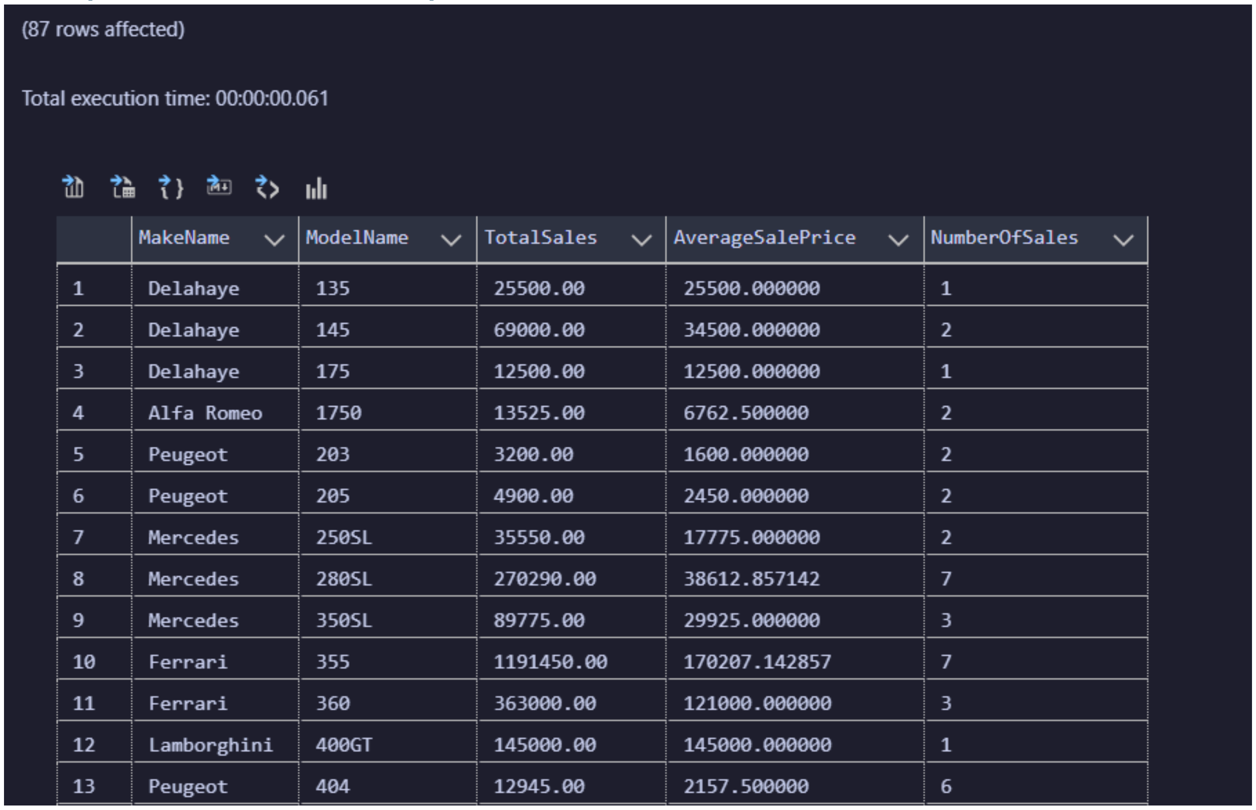
Order by :

Table Name	Column Name	Sort Order
N/A	N/A	N/A

Problem solving Query:

```
USE PrestigeCars;

GO


-- CTE to aggregate sales by Make and Model

WITH CarSalesSummary AS (

    SELECT

        MK.MakeName,

        MD.ModelName,

        SUM(SD.SalePrice) AS TotalSales,

        AVG(SD.SalePrice) AS AverageSalePrice,

        COUNT(SD.SalesID) AS NumberOfSales

    FROM Data.Sales SA

    INNER JOIN Data.SalesDetails SD ON SA.SalesID = SD.SalesID

    INNER JOIN Data.Stock ST ON SD.StockID = ST.StockCode

    INNER JOIN Data.Model MD ON ST.ModelID = MD.ModelID

    INNER JOIN Data.Make MK ON MD.MakeID = MK.MakeID

    GROUP BY MK.MakeName, MD.ModelName
)
-- Select the data in relational format
SELECT MakeName, ModelName, TotalSales, AveragesalePrice, NumberOfSales
FROM CarSalesSummary


FOR JSON PATH, ROOT('CarSalesSummary');
```

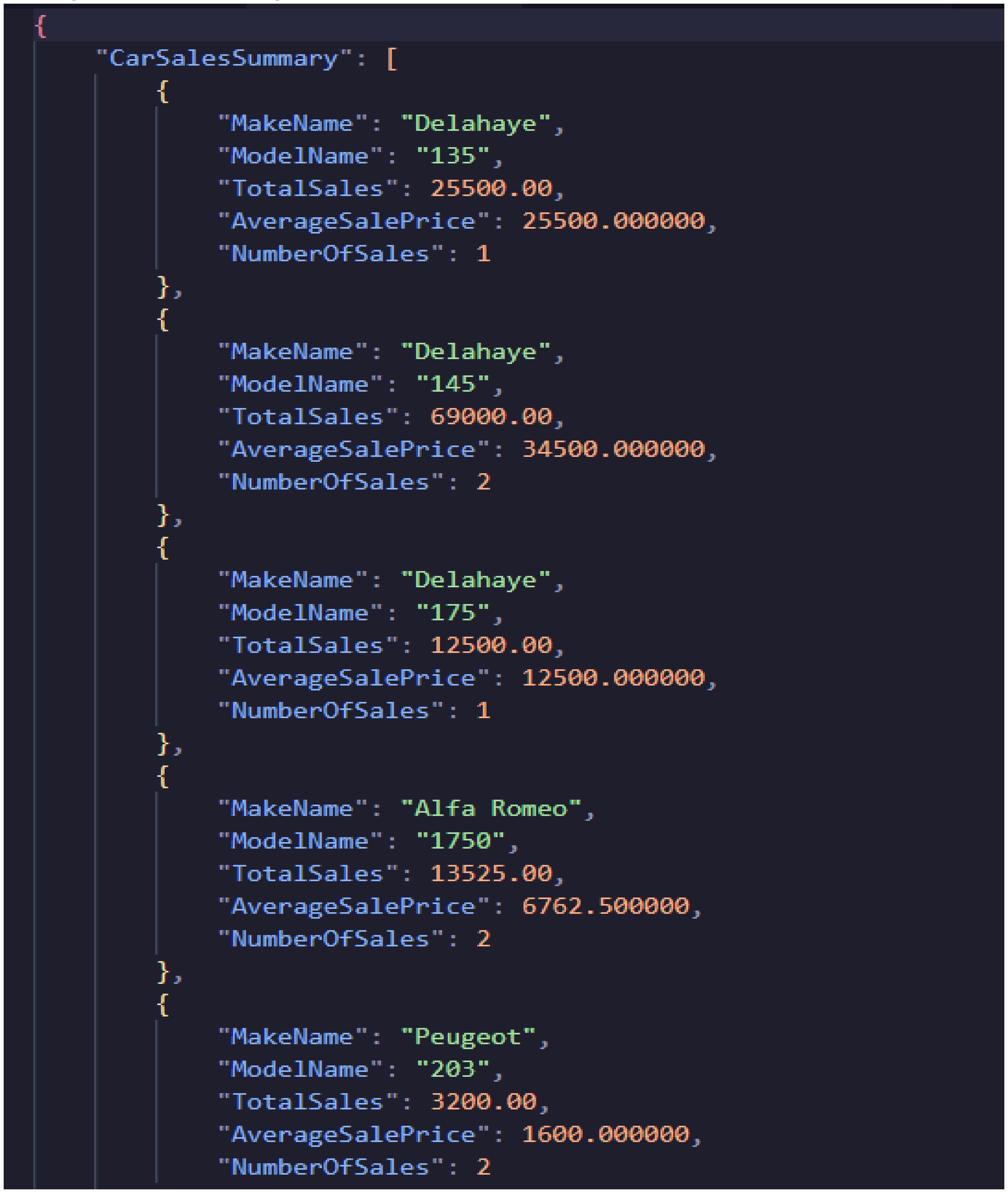
Sample Relational Output with total number of rows returned:



```
(87 rows affected)

Total execution time: 00:00:00.061
```

	MakeName	ModelName	TotalSales	AverageSalePrice	NumberOfSales
1	Delahaye	135	25500.00	25500.000000	1
2	Delahaye	145	69000.00	34500.000000	2
3	Delahaye	175	12500.00	12500.000000	1
4	Alfa Romeo	1750	13525.00	6762.500000	2
5	Peugeot	203	3200.00	1600.000000	2
6	Peugeot	205	4900.00	2450.000000	2
7	Mercedes	250SL	35550.00	17775.000000	2
8	Mercedes	280SL	270290.00	38612.857142	7
9	Mercedes	350SL	89775.00	29925.000000	3
10	Ferrari	355	1191450.00	170207.142857	7
11	Ferrari	360	363000.00	121000.000000	3
12	Lamborghini	400GT	145000.00	145000.000000	1
13	Peugeot	404	12945.00	2157.500000	6

Sample JSON Output with total number of rows returned:



```
{
    "CarSalesSummary": [
        {
            "MakeName": "Delahaye",
            "ModelName": "135",
            "TotalSales": 25500.00,
            "AverageSalePrice": 25500.000000,
            "NumberOfSales": 1
        },
        {
            "MakeName": "Delahaye",
            "ModelName": "145",
            "TotalSales": 69000.00,
            "AverageSalePrice": 34500.000000,
            "NumberOfSales": 2
        },
        {
            "MakeName": "Delahaye",
            "ModelName": "175",
            "TotalSales": 12500.00,
            "AverageSalePrice": 12500.000000,
            "NumberOfSales": 1
        },
        {
            "MakeName": "Alfa Romeo",
            "ModelName": "1750",
            "TotalSales": 13525.00,
            "AverageSalePrice": 6762.500000,
            "NumberOfSales": 2
        },
        {
            "MakeName": "Peugeot",
            "ModelName": "203",
            "TotalSales": 3200.00,
            "AverageSalePrice": 1600.000000,
            "NumberOfSales": 2
```

WORST QUERY (1)

USE PrestigeCars

NOTE DATABASE SCRIPT DOES NOT DEFINE FOREIGN KEY AND PRIMARY KEY RELATIONSHIP

CANNOT AN ER DIAGRAM FROM A VIEW

Columns from their respective tables in the select clause:

Table Name	Column Name
[Data].[SalesByCountry]- VIEW	CountryName, SalePrice, LineItemDiscount, InvoiceNumber

Order by :

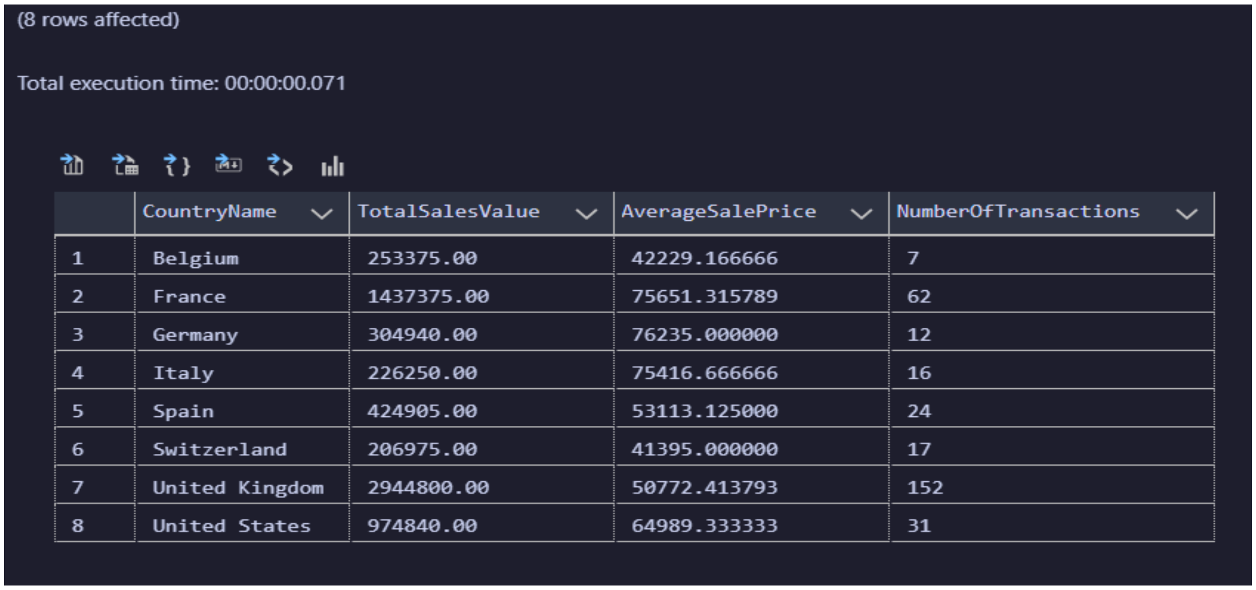
Table Name	Column Name	Sort Order
N/A	N/A	N/A

Problem solving Query:

```
USE PrestigeCars;

GO


-- Query using the view [Data].[SalesByCountry] to summarize sales by country

WITH SalesSummary AS (

    SELECT

        CountryName,

        SUM(SalePrice - LineItemDiscount) AS TotalSalesValue,

        AVG(SalePrice - LineItemDiscount) AS AverageSalePrice,

        COUNT(DISTINCT InvoiceNumber) AS NumberOfTransactions

    FROM [Data].[SalesByCountry]

    GROUP BY CountryName

)

SELECT
```

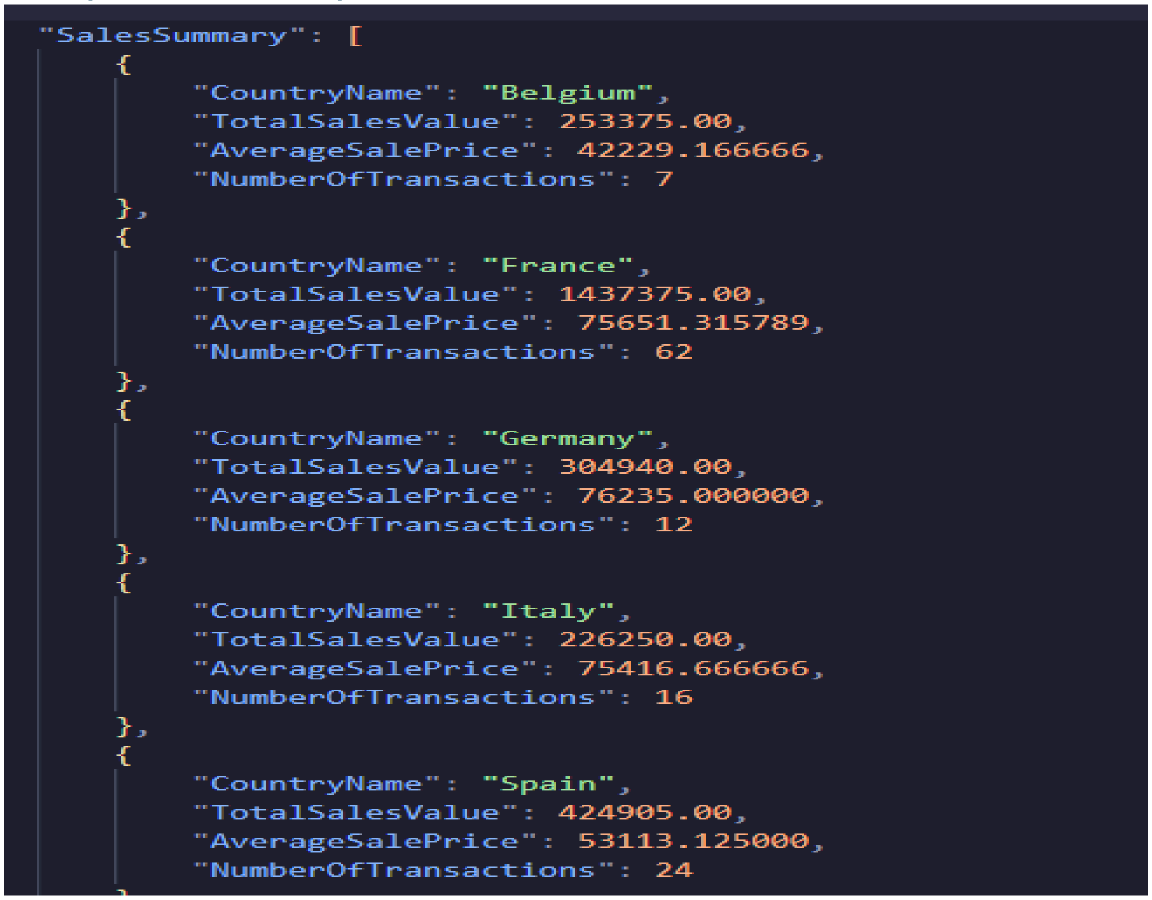
```
    CountryName,
    TotalSalesValue,
    AverageSalePrice,
    NumberOfTransactions
FROM SalesSummary

-- FOR JSON PATH, ROOT('SalesSummary');
```

Sample Relational Output with total number of rows returned:



```
(8 rows affected)

Total execution time: 00:00:00.071

    CountryName       TotalSalesValue   AverageSalePrice   NumberOfTransactions
1   Belgium           253375.00         42229.166666       7
2   France            1437375.00        75651.315789       62
3   Germany           304940.00         76235.000000       12
4   Italy             226250.00         75416.666666       16
5   Spain             424905.00         53113.125000       24
6   Switzerland       206975.00         41395.000000       17
7   United Kingdom    2944800.00        50772.413793       152
8   United States     974840.00         64989.333333       31
```

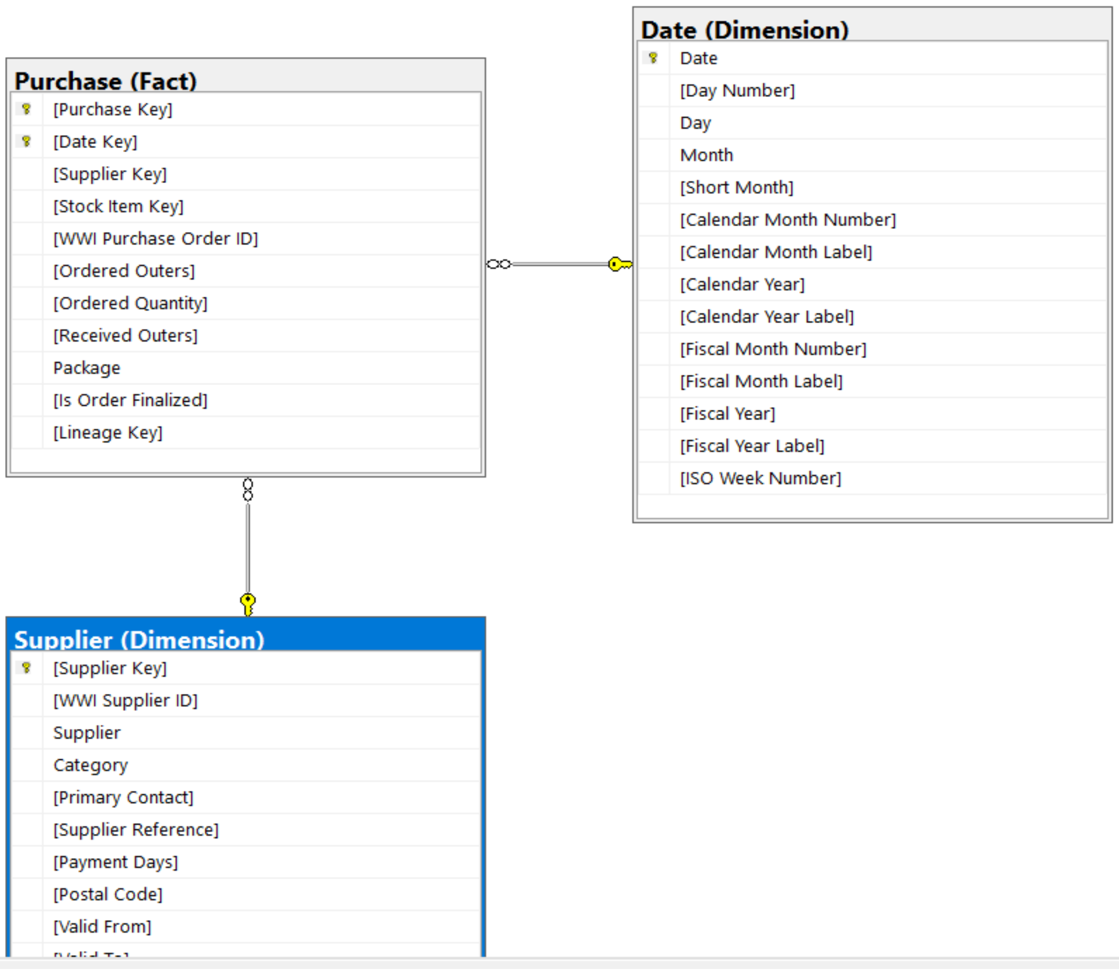
Sample JSON Output with total number of rows returned:



```
"SalesSummary": [
    {
        "CountryName": "Belgium",
        "TotalSalesValue": 253375.00,
        "AverageSalePrice": 42229.166666,
        "NumberOfTransactions": 7
    },
    {
        "CountryName": "France",
        "TotalSalesValue": 1437375.00,
        "AverageSalePrice": 75651.315789,
        "NumberOfTransactions": 62
    },
    {
        "CountryName": "Germany",
        "TotalSalesValue": 304940.00,
        "AverageSalePrice": 76235.000000,
        "NumberOfTransactions": 12
    },
    {
        "CountryName": "Italy",
        "TotalSalesValue": 226250.00,
        "AverageSalePrice": 75416.666666,
        "NumberOfTransactions": 16
    },
    {
        "CountryName": "Spain",
        "TotalSalesValue": 424905.00,
        "AverageSalePrice": 53113.125000,
        "NumberOfTransactions": 24
```

WORST QUERY (2)

Use: WideWorldImportersDW



PROPOSITION: Formulate or update a query in the WideWorldImportersDW database to compute and summarize monthly purchases from suppliers. This involves calculating both the total and average quantity of items ordered from each supplier, segmented by month and year.

Columns from their respective tables in the select clause:

Table Name	Column Name
[Data].[SalesByCountry]- VIEW	CountryName, SalePrice, LineItemDiscount, InvoiceNumber
Fact.Purchase	Date Key, Supplier Key
Dimension.Date	Date
Dimension.Supplier	Supplier Key

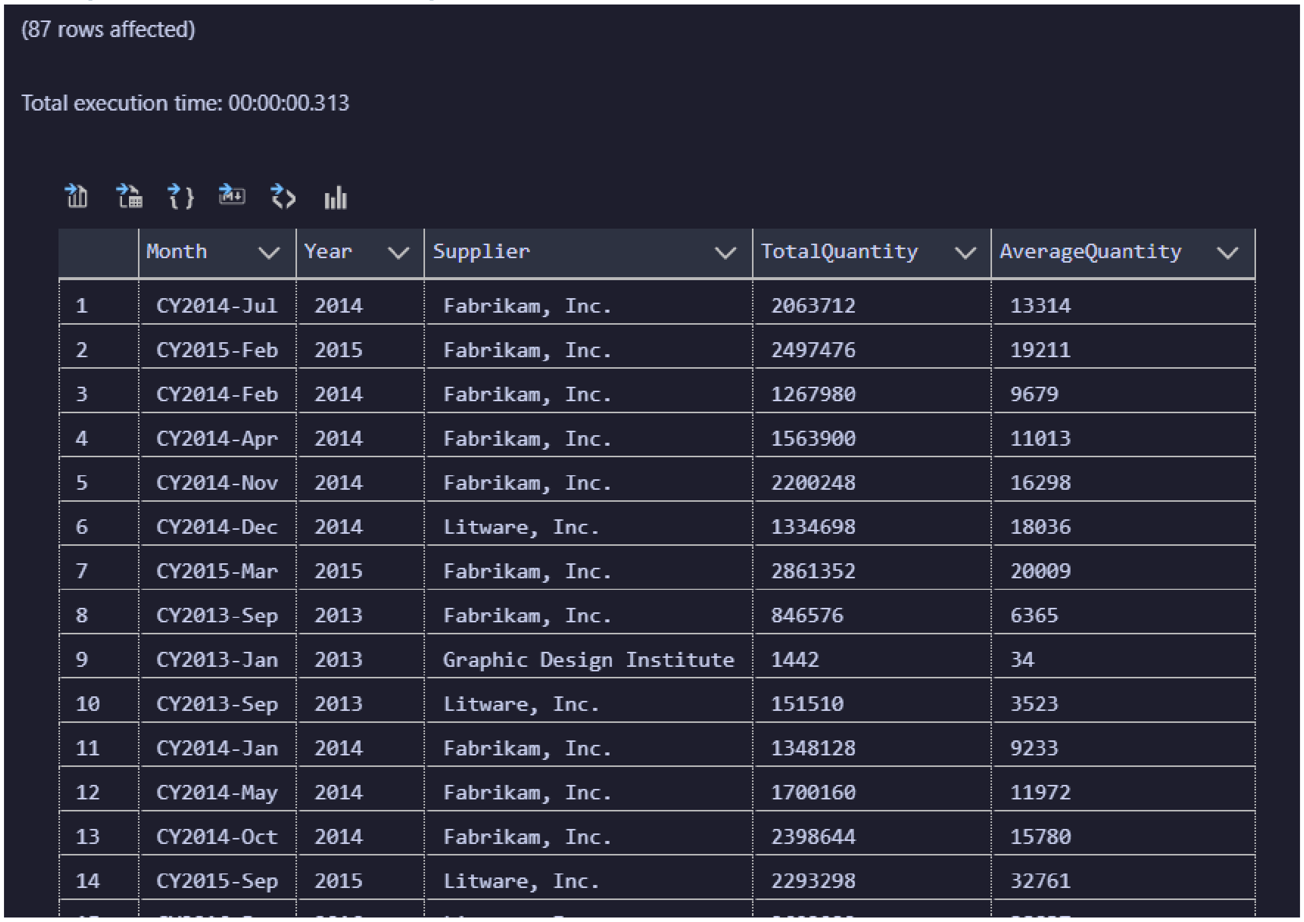
Order by :

Table Name	Column Name	Sort Order
SalesSummary CTE	TotalPurchaseAmount	Descending

Problem solving Query:

```
Use WideWorldImportersDW

;WITH MonthlySupplierPurchases AS (

    SELECT

        d.[Calendar Month Label] AS Month,

        d.[Calendar Year] AS Year,

        s.Supplier,

        SUM(p.[Ordered Quantity]) AS TotalQuantity,

        AVG(p.[Ordered Quantity]) AS AverageQuantity

    FROM

        Fact.Purchase AS p

        JOIN Dimension.Date AS d ON p.[Date Key] = d.Date

        JOIN Dimension.Supplier AS s ON p.[Supplier Key] = s.[Supplier Key]

    GROUP BY

        d.[Calendar Month Label],

        d.[Calendar Year],
        s.Supplier
)
-- Relational output
SELECT Month, Year, Supplier, TotalQuantity, AverageQuantity
FROM MonthlySupplierPurchases


FOR JSON PATH, ROOT('MonthlySupplierPurchases');
```

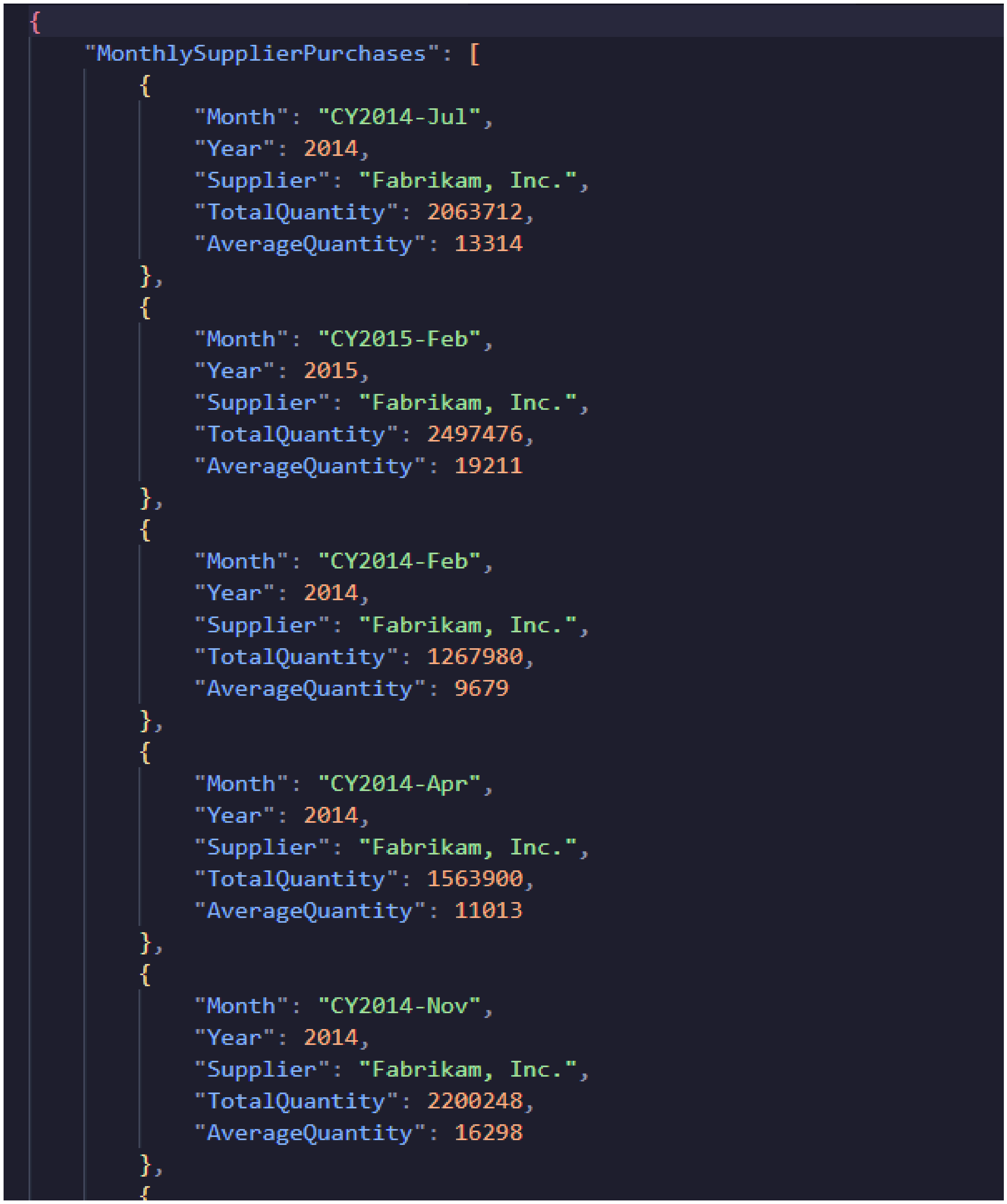
Sample Relational Output with total number of rows returned:



```
(87 rows affected)

Total execution time: 00:00:00.313
```

	Month	Year	Supplier	TotalQuantity	AverageQuantity
1	CY2014-Jul	2014	Fabrikam, Inc.	2063712	13314
2	CY2015-Feb	2015	Fabrikam, Inc.	2497476	19211
3	CY2014-Feb	2014	Fabrikam, Inc.	1267980	9679
4	CY2014-Apr	2014	Fabrikam, Inc.	1563900	11013
5	CY2014-Nov	2014	Fabrikam, Inc.	2200248	16298
6	CY2014-Dec	2014	Litware, Inc.	1334698	18036
7	CY2015-Mar	2015	Fabrikam, Inc.	2861352	20009
8	CY2013-Sep	2013	Fabrikam, Inc.	846576	6365
9	CY2013-Jan	2013	Graphic Design Institute	1442	34
10	CY2013-Sep	2013	Litware, Inc.	151510	3523
11	CY2014-Jan	2014	Fabrikam, Inc.	1348128	9233
12	CY2014-May	2014	Fabrikam, Inc.	1700160	11972
13	CY2014-Oct	2014	Fabrikam, Inc.	2398644	15780
14	CY2015-Sep	2015	Litware, Inc.	2293298	32761

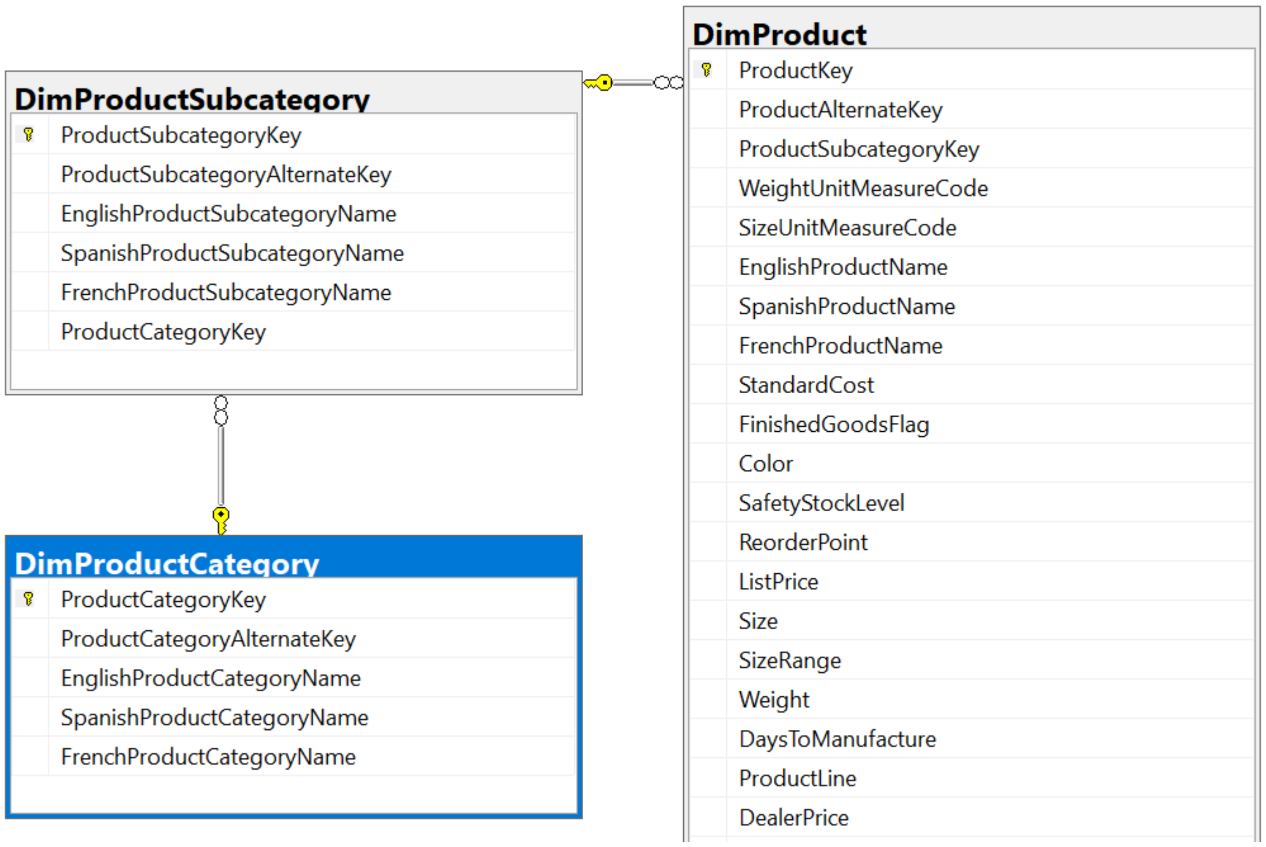
Sample JSON Output with total number of rows returned:



```
{
    "MonthlySupplierPurchases": [
        {
            "Month": "CY2014-Jul",
            "Year": 2014,
            "Supplier": "Fabrikam, Inc.",
            "TotalQuantity": 2063712,
            "AverageQuantity": 13314
        },
        {
            "Month": "CY2015-Feb",
            "Year": 2015,
            "Supplier": "Fabrikam, Inc.",
            "TotalQuantity": 2497476,
            "AverageQuantity": 19211
        },
        {
            "Month": "CY2014-Feb",
            "Year": 2014,
            "Supplier": "Fabrikam, Inc.",
            "TotalQuantity": 1267980,
            "AverageQuantity": 9679
        },
        {
            "Month": "CY2014-Apr",
            "Year": 2014,
            "Supplier": "Fabrikam, Inc.",
            "TotalQuantity": 1563900,
            "AverageQuantity": 11013
        },
        {
            "Month": "CY2014-Nov",
            "Year": 2014,
            "Supplier": "Fabrikam, Inc.",
            "TotalQuantity": 2200248,
            "AverageQuantity": 16298
        },
        {
```

WORST QUERY (3)

USE AdeventureWorksDW2017



Proposition:

Utilizing a Common Table Expression (CTE) named ProductPriceSummary, the query systematically aggregates product pricing data from the dbo.DimProduct table, alongside hierarchical categorization from dbo.DimProductSubcategory and dbo.DimProductCategory tables. By filtering out products without a list price, it calculates the average list price within each product category and subcategory. The results are grouped accordingly, offering a refined view that facilitates understanding of average pricing across various product segments. This strategic grouping provides valuable insights for pricing analysis, marketing strategies, and product placement decisions.

Columns from their respective tables in the select clause:

Table Name	Column Name
dbo.DimProduct (p)	ListPrice, ProductSubcategoryKey
dbo.DimProductSubcategory (psc)	EnglishProductSubcategoryName, ProductCategoryKey
dbo.DimProductCategory (pc)	EnglishProductCategoryName

Order by :

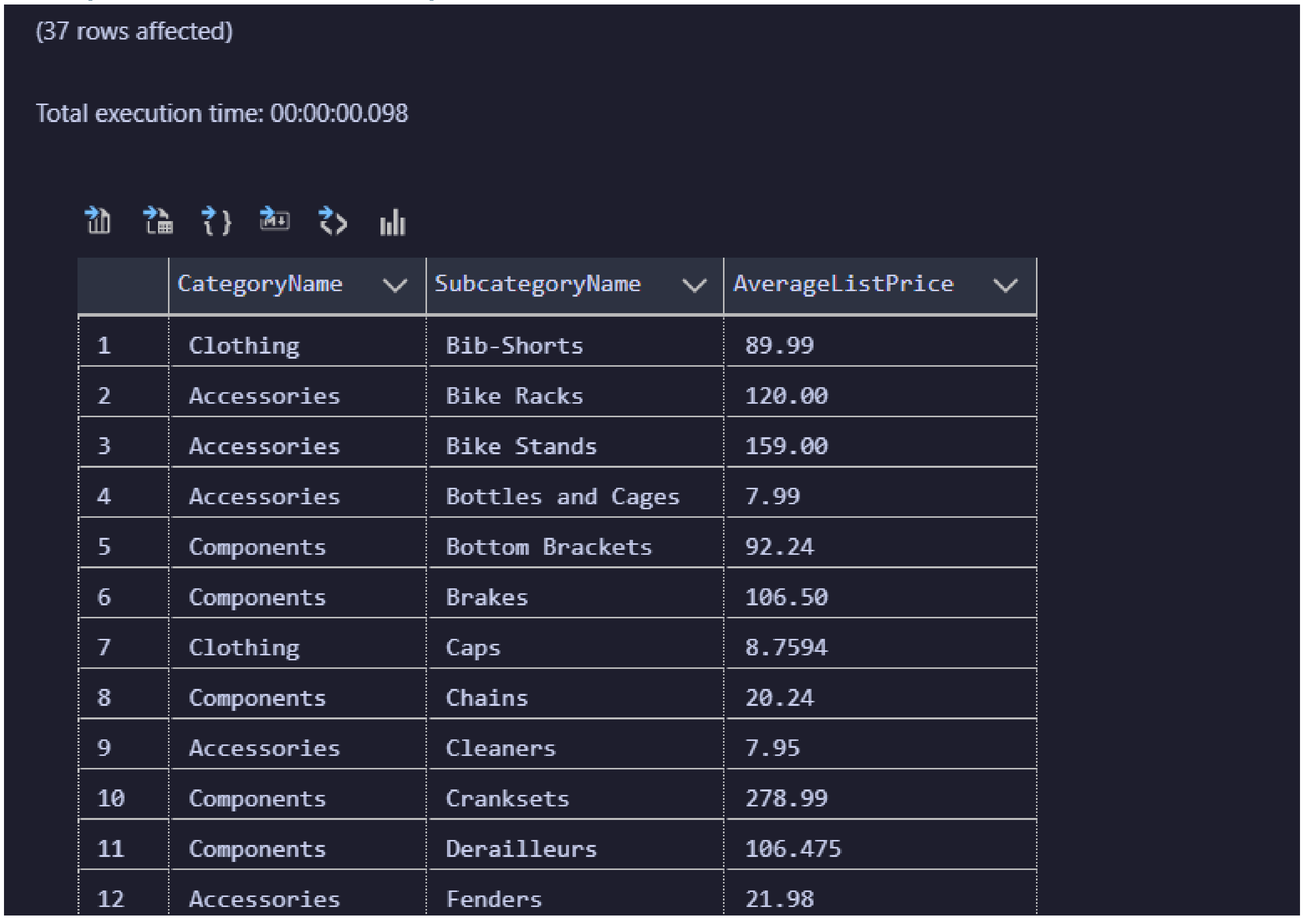
Table Name	Column Name	Sort Order
SalesSummary CTE	TotalPurchaseAmount	Descending

Problem solving Query:

```
Use AdventureWorksDW2017


-- Calculating average list price by product category and subcategory

;WITH ProductPriceSummary AS (

    SELECT

        pc.EnglishProductCategoryName AS CategoryName,

        psc.EnglishProductSubcategoryName AS SubcategoryName,

        AVG(p.ListPrice) AS AverageListPrice

    FROM dbo.DimProduct AS p

    INNER JOIN dbo.DimProductSubcategory AS psc ON p.ProductSubcategoryKey =
psc.ProductSubcategoryKey

    INNER JOIN dbo.DimProductCategory AS pc ON psc.ProductCategoryKey =
pc.ProductCategoryKey

    WHERE p.ListPrice > 0 -- Excluding products with no list price

    GROUP BY pc.EnglishProductCategoryName, psc.EnglishProductSubcategoryName

)

-- Relational output
```

```
SELECT CategoryName, SubcategoryName, AverageListPrice

FROM ProductPriceSummary


-- JSON output

FOR JSON PATH, ROOT('ProductPriceSummary');
```

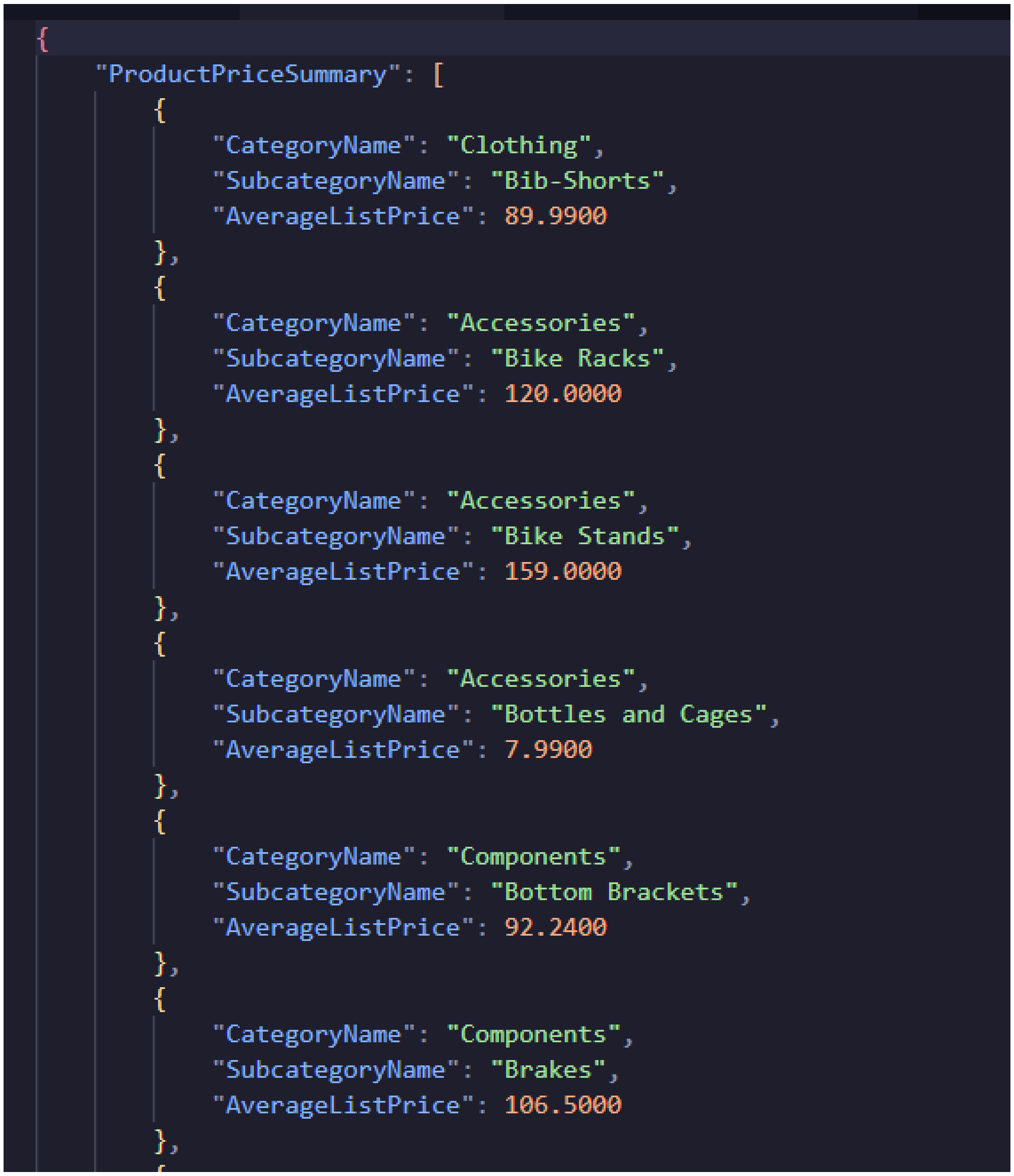
Sample Relational Output with total number of rows returned:



```
(37 rows affected)

Total execution time: 00:00:00.098
```

	CategoryName	SubcategoryName	AverageListPrice
1	Clothing	Bib-Shorts	89.99
2	Accessories	Bike Racks	120.00
3	Accessories	Bike Stands	159.00
4	Accessories	Bottles and Cages	7.99
5	Components	Bottom Brackets	92.24
6	Components	Brakes	106.50
7	Clothing	Caps	8.7594
8	Components	Chains	20.24
9	Accessories	Cleaners	7.95
10	Components	Cranksets	278.99
11	Components	Derailleurs	106.475
12	Accessories	Fenders	21.98

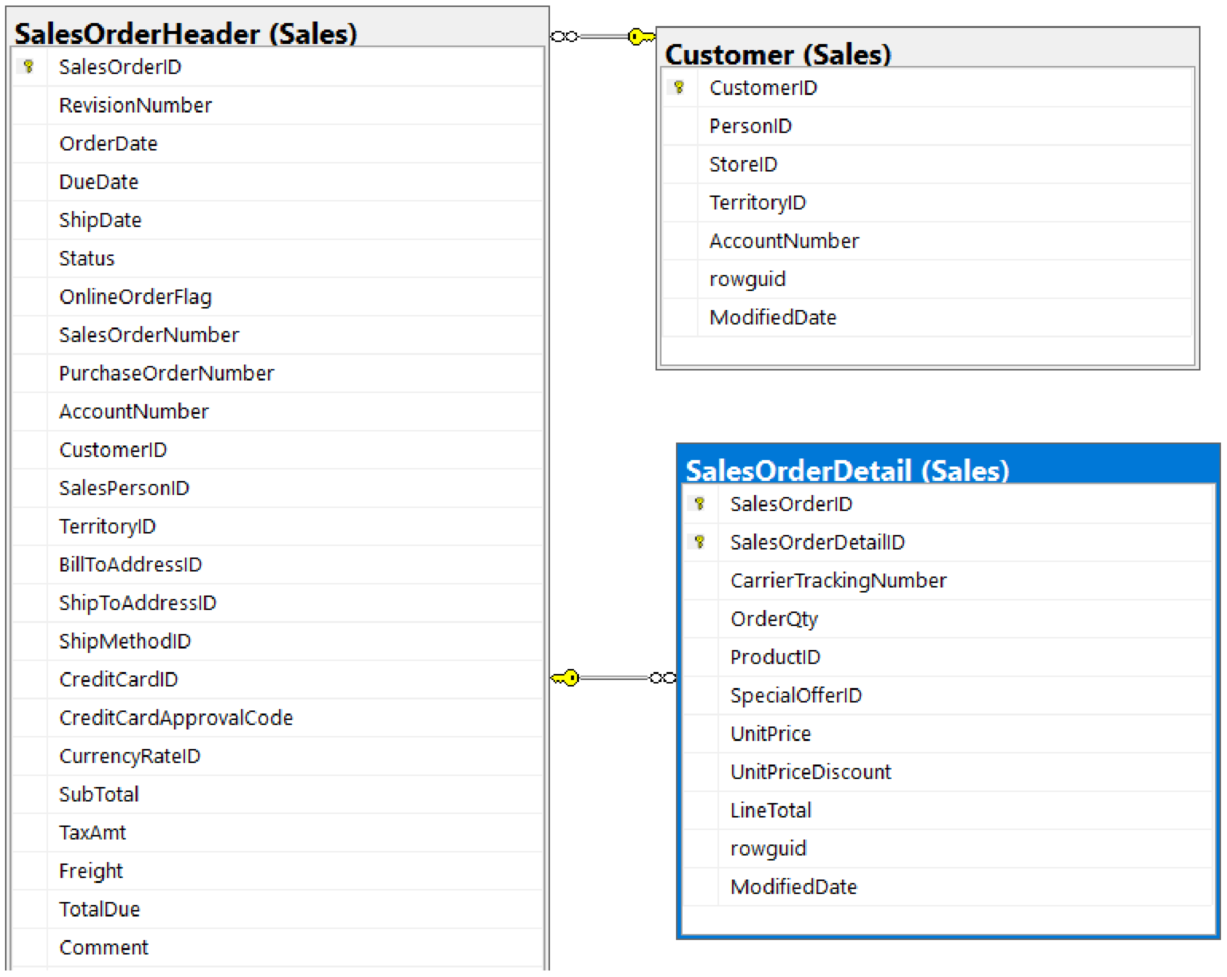
Sample JSON Output with total number of rows returned:



```
{
    "ProductPriceSummary": [
        {
            "CategoryName": "Clothing",
            "SubcategoryName": "Bib-Shorts",
            "AverageListPrice": 89.9900
        },
        {
            "CategoryName": "Accessories",
            "SubcategoryName": "Bike Racks",
            "AverageListPrice": 120.0000
        },
        {
            "CategoryName": "Accessories",
            "SubcategoryName": "Bike Stands",
            "AverageListPrice": 159.0000
        },
        {
            "CategoryName": "Accessories",
            "SubcategoryName": "Bottles and Cages",
            "AverageListPrice": 7.9900
        },
        {
            "CategoryName": "Components",
            "SubcategoryName": "Bottom Brackets",
            "AverageListPrice": 92.2400
        },
        {
            "CategoryName": "Components",
            "SubcategoryName": "Brakes",
            "AverageListPrice": 106.5000
        },
```

MEDIUM QUERY

USE AdeventureWorks2017



Proposition: Design or refine a query in the AdventureWorks2017 database for the purpose of summarizing sales data by customer. This summary includes the total number of orders, the total sales value, and the average value of orders placed by each customer.

Columns from their respective tables in the select clause:

Table Name	Column Name
Sales.Customer (c)	CustomerID
Sales.SalesOrderHeader (soh)	CustomerID, SalesOrderID
Sales.SalesOrderDetail (sod)	SalesOrderID, LineTotal

Order by :

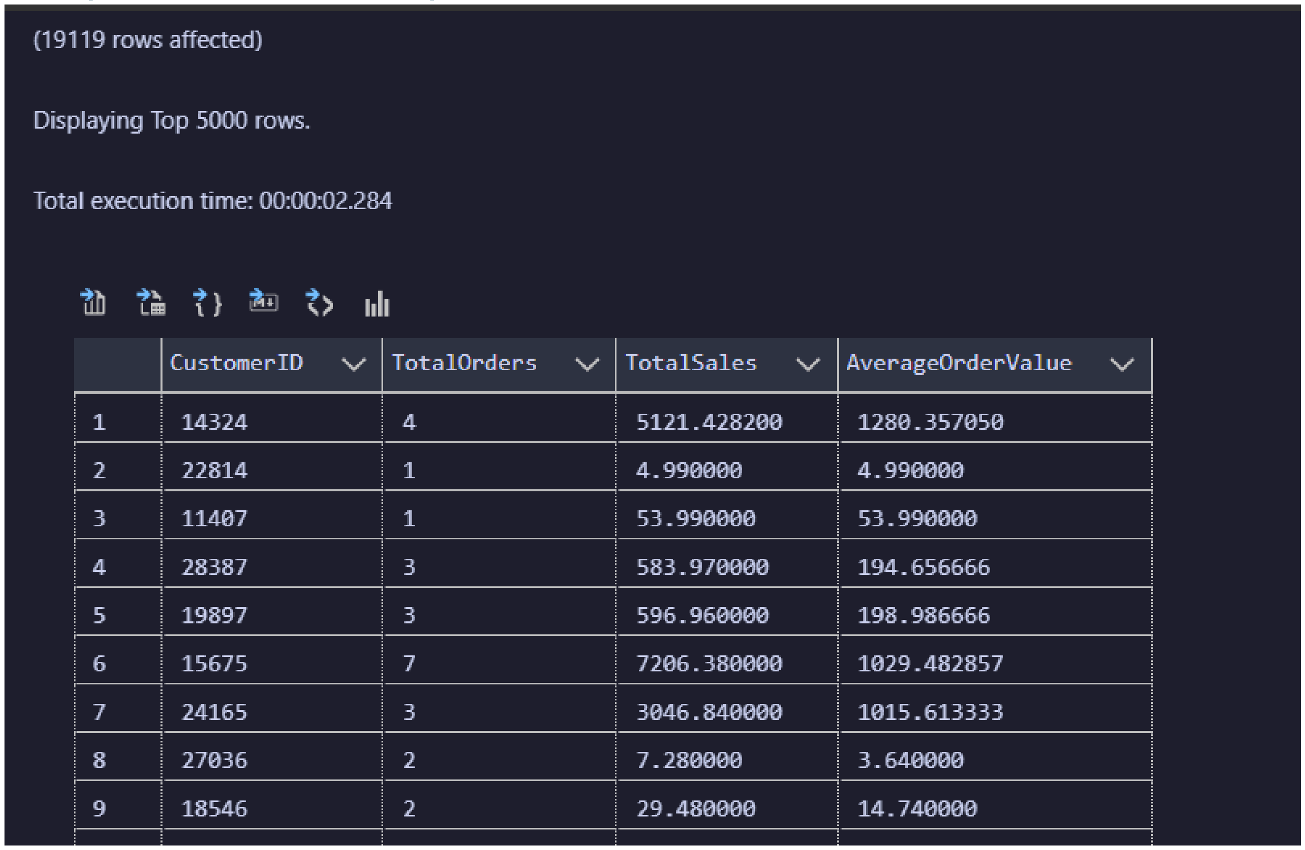
Table Name	Column Name	Sort Order
Sales.Customer	CustomerID	Descending

Problem solving Query:

```
Use AdventureWorks2017

;WITH CustomerSalesSummary AS (

    SELECT

        c.CustomerID,

        COUNT(soh.SalesOrderID) AS TotalOrders,

        SUM(sod.LineTotal) AS TotalSales,

        AVG(sod.LineTotal) AS AverageOrderValue

    FROM Sales.Customer c

    INNER JOIN Sales.SalesOrderHeader soh ON c.CustomerID = soh.CustomerID

    INNER JOIN Sales.SalesOrderDetail sod ON soh.SalesOrderID = sod.SalesOrderID

    GROUP BY c.CustomerID

)

-- Relational output

SELECT CustomerID, TotalOrders, TotalSales, AverageOrderValue
```

```
FROM CustomerSalesSummary


-- JSON output

FOR JSON PATH, ROOT('CustomerSalesSummary');
```

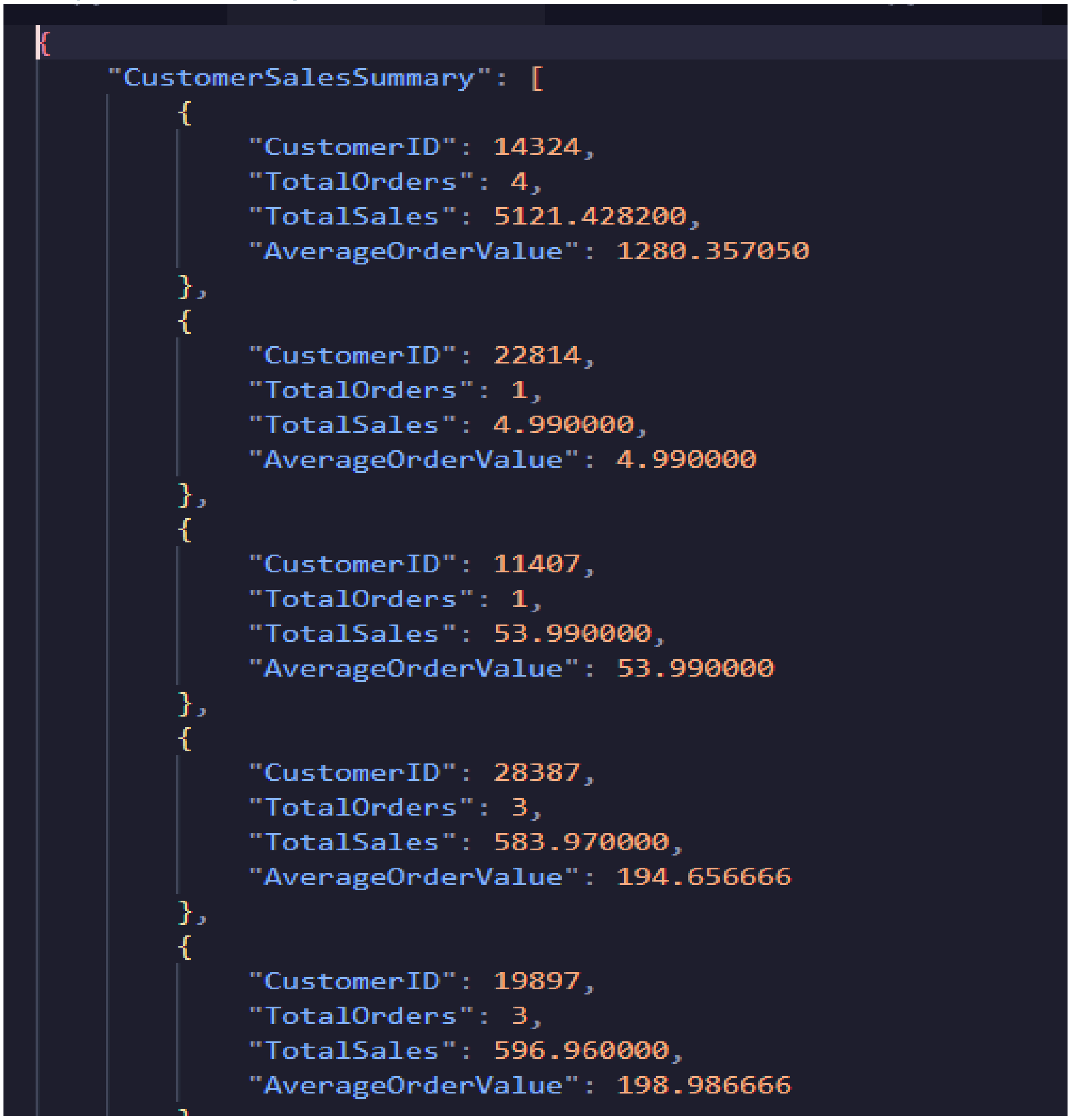
Sample Relational Output with total number of rows returned:



```
(19119 rows affected)

Displaying Top 5000 rows.

Total execution time: 00:00:02.284
```

	CustomerID	TotalOrders	TotalSales	AverageOrderValue
1	14324	4	5121.428200	1280.357050
2	22814	1	4.990000	4.990000
3	11407	1	53.990000	53.990000
4	28387	3	583.970000	194.656666
5	19897	3	596.960000	198.986666
6	15675	7	7206.380000	1029.482857
7	24165	3	3046.840000	1015.613333
8	27036	2	7.280000	3.640000
9	18546	2	29.480000	14.740000

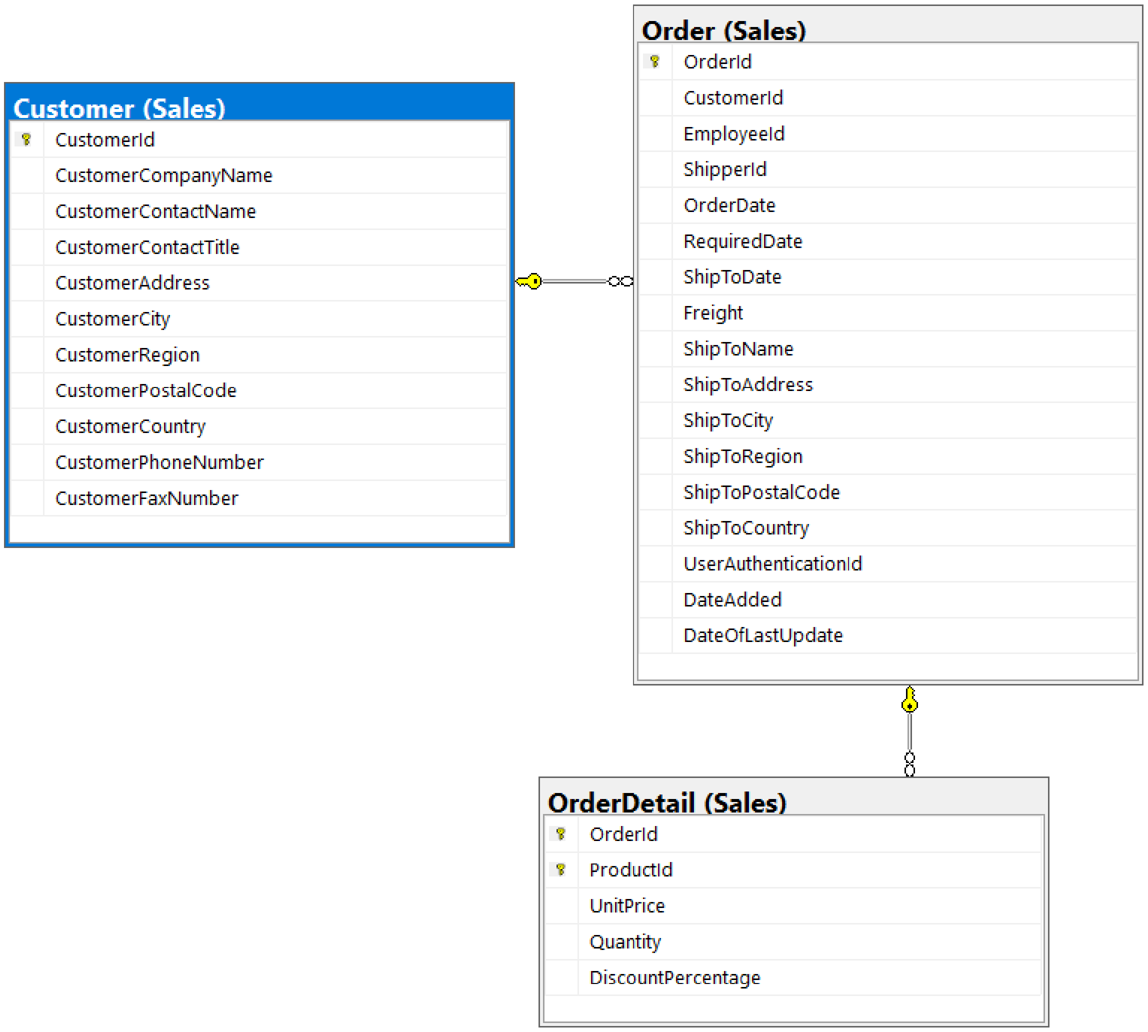
Sample JSON Output with total number of rows returned:



```
{
    "CustomerSalesSummary": [
        {
            "CustomerID": 14324,
            "TotalOrders": 4,
            "TotalSales": 5121.428200,
            "AverageOrderValue": 1280.357050
        },
        {
            "CustomerID": 22814,
            "TotalOrders": 1,
            "TotalSales": 4.990000,
            "AverageOrderValue": 4.990000
        },
        {
            "CustomerID": 11407,
            "TotalOrders": 1,
            "TotalSales": 53.990000,
            "AverageOrderValue": 53.990000
        },
        {
            "CustomerID": 28387,
            "TotalOrders": 3,
            "TotalSales": 583.970000,
            "AverageOrderValue": 194.656666
        },
        {
            "CustomerID": 19897,
            "TotalOrders": 3,
            "TotalSales": 596.960000,
            "AverageOrderValue": 198.986666
```

MEDIUM QUERY

Use Northwinds2022TSQLV7



PROPOSITION: Develop or enhance a query within the AdventureWorks2017 database to aggregate sales data by customer. This query focuses on computing the total number of orders, the total sales value, and the average order value for each customer.

Columns from their respective tables in the select clause:

Table Name	Column Name
Sales.Customer (c)	CustomerCompanyName
Sales.[Order] (o)	CustomerId, OrderId
Sales.OrderDetail (od)	OrderId, UnitPrice, Quantity

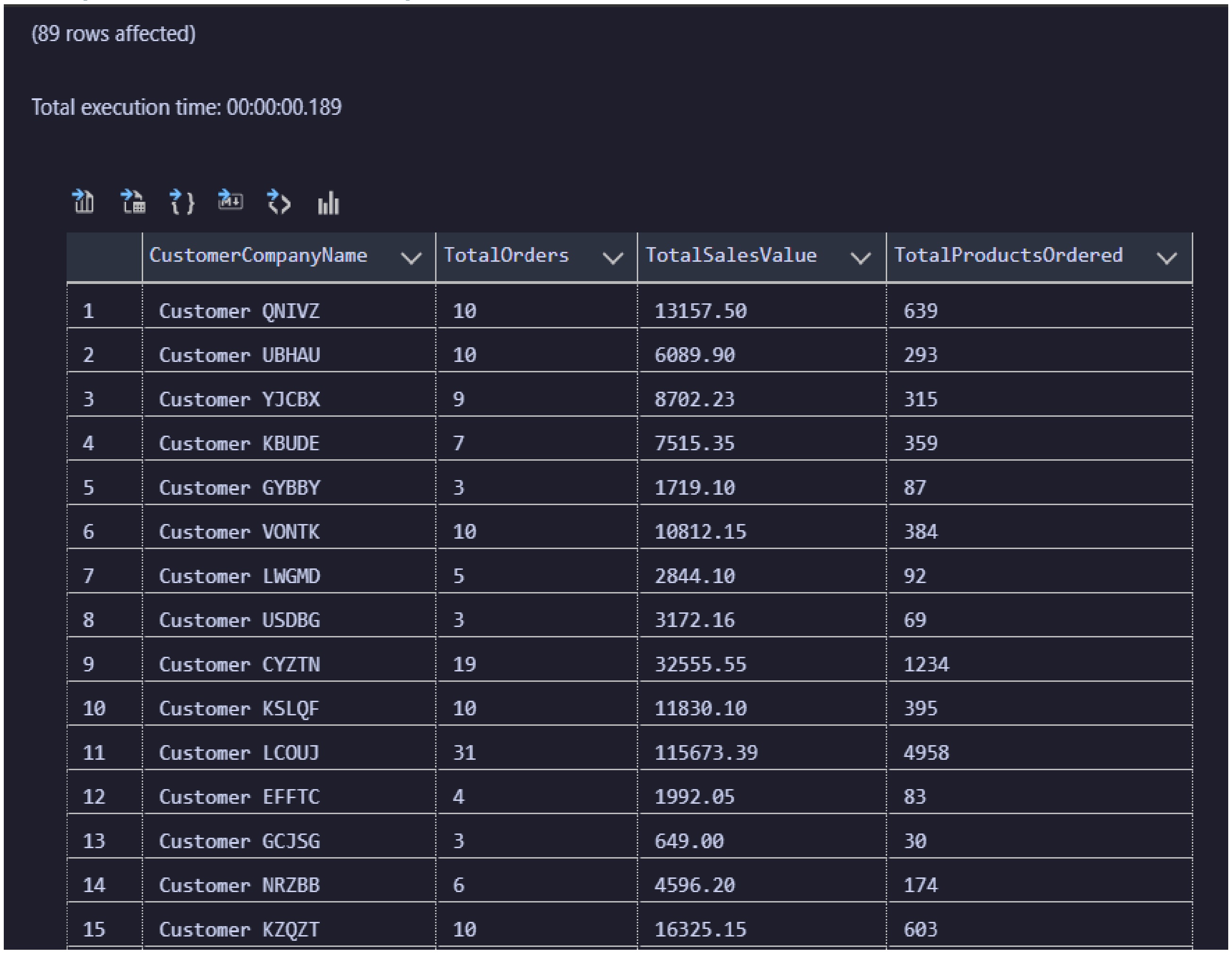
Order by :

Table Name	Column Name	Sort Order
N/A	N/A	N/A

Problem solving Query:

```
Use Northwinds2022TSQLV7

;WITH CustomerOrderSummary AS (

    SELECT

        c.CustomerCompanyName,

        COUNT(DISTINCT o.OrderId) AS TotalOrders,

        SUM(od.UnitPrice * od.Quantity) AS TotalSalesValue,

        SUM(od.Quantity) AS TotalProductsOrdered

    FROM Sales.[Order] o

    INNER JOIN Sales.Customer c ON o.CustomerId = c.CustomerId

    INNER JOIN Sales.OrderDetail od ON o.OrderId = od.OrderId

    GROUP BY c.CustomerCompanyName

)

-- Relational output

SELECT CustomerCompanyName, TotalOrders, TotalSalesValue, TotalProductsOrdered

FROM CustomerOrderSummary


-- JSON output

 FOR JSON PATH, ROOT('CustomerOrderSummary');
```

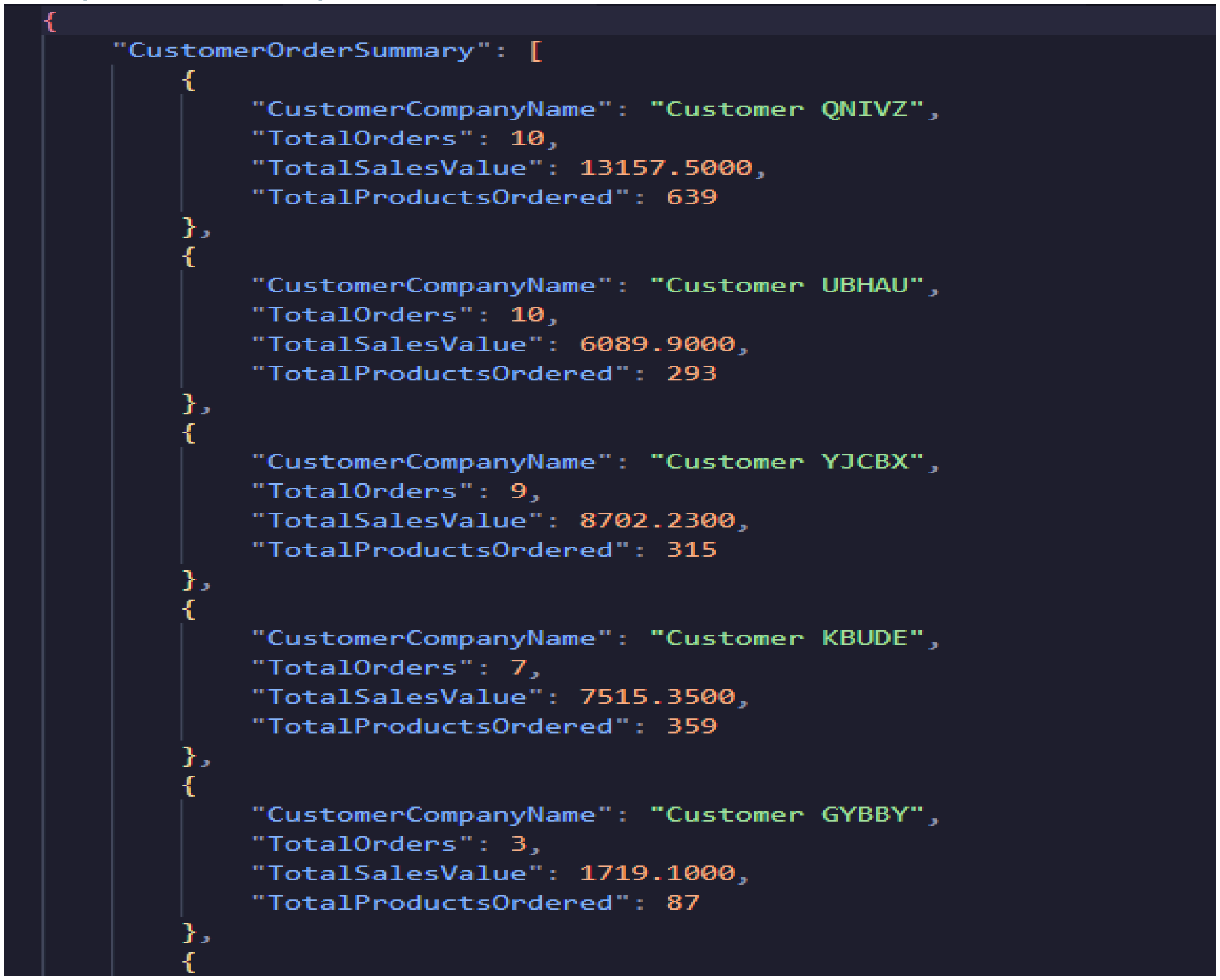
Sample Relational Output with total number of rows returned:



```
(89 rows affected)

Total execution time: 00:00:00.189
```

	CustomerCompanyName	TotalOrders	TotalSalesValue	TotalProductsOrdered
1	Customer QNIVZ	10	13157.50	639
2	Customer UBHAU	10	6089.90	293
3	Customer YJCBX	9	8702.23	315
4	Customer KBUDE	7	7515.35	359
5	Customer GYBBY	3	1719.10	87
6	Customer VONTK	10	10812.15	384
7	Customer LWGMD	5	2844.10	92
8	Customer USDBG	3	3172.16	69
9	Customer CYZTN	19	32555.55	1234
10	Customer KSLQF	10	11830.10	395
11	Customer LCOUJ	31	115673.39	4958
12	Customer EFFTC	4	1992.05	83
13	Customer GCJSG	3	649.00	30
14	Customer NRZBB	6	4596.20	174
15	Customer KZQZT	10	16325.15	603

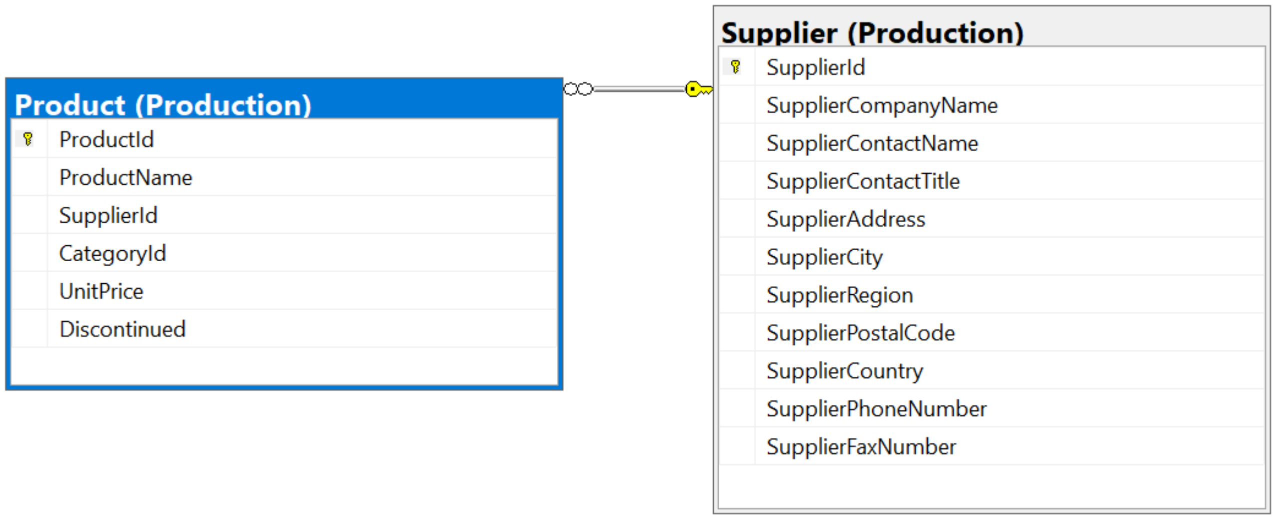
Sample JSON Output with total number of rows returned:



```
{
    "CustomerOrderSummary": [
        {
            "CustomerCompanyName": "Customer QNIVZ",
            "TotalOrders": 10,
            "TotalSalesValue": 13157.5000,
            "TotalProductsOrdered": 639
        },
        {
            "CustomerCompanyName": "Customer UBHAU",
            "TotalOrders": 10,
            "TotalSalesValue": 6089.9000,
            "TotalProductsOrdered": 293
        },
        {
            "CustomerCompanyName": "Customer YJCBX",
            "TotalOrders": 9,
            "TotalSalesValue": 8702.2300,
            "TotalProductsOrdered": 315
        },
        {
            "CustomerCompanyName": "Customer KBUDE",
            "TotalOrders": 7,
            "TotalSalesValue": 7515.3500,
            "TotalProductsOrdered": 359
        },
        {
            "CustomerCompanyName": "Customer GYBBY",
            "TotalOrders": 3,
            "TotalSalesValue": 1719.1000,
            "TotalProductsOrdered": 87
        },
        {
```

MEDIUM QUERY

Use Northwinds2022TSQLV7



PREPOSITION: Design or modify a query in the Northwinds2022TSQLV7 database to compile a summary of orders by customer company. This query aims to detail the total number of distinct orders, the aggregate sales value, and the total quantity of products ordered, all categorized by customer company name.

Columns from their respective tables in the select clause:

Table Name	Column Name
Production.Supplier:	SupplierCompanyName
Production.Product	ProductId SupplierId UnitPrice

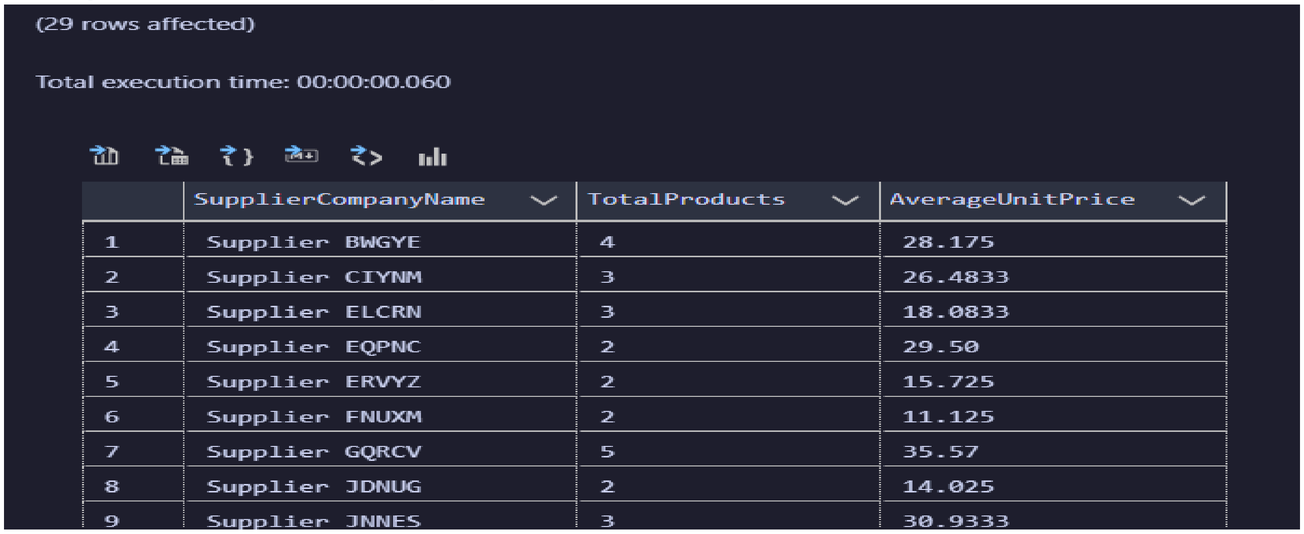
Order by :

Table Name	Column Name	Sort Order
N/A	N/A	N/A

Problem solving Query:

```
Use Northwinds2022TSQLV7

;WITH SupplierProductSummary AS (

    SELECT

        s.SupplierCompanyName,

        COUNT(p.ProductId) AS TotalProducts,

        AVG(p.UnitPrice) AS AverageUnitPrice

    FROM Production.Product p

    INNER JOIN Production.Supplier s ON p.SupplierId = s.SupplierId

    GROUP BY s.SupplierCompanyName

)

-- Relational output

SELECT SupplierCompanyName, TotalProducts, AverageUnitPrice

FROM SupplierProductSummary


-- -- JSON output


FOR JSON PATH, ROOT('SupplierProductSummary');
```

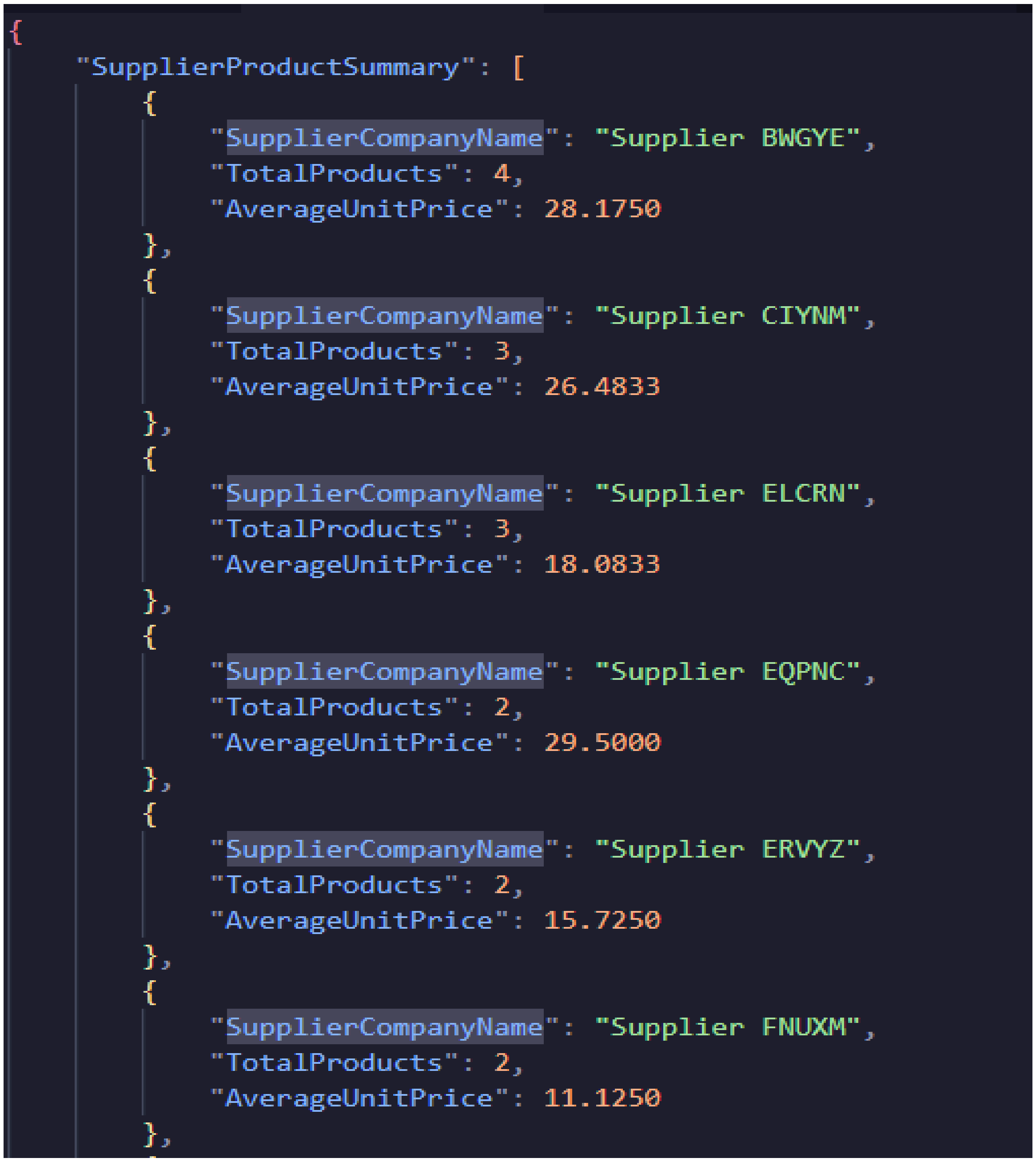
Sample Relational Output with total number of rows returned:



```
(29 rows affected)

Total execution time: 00:00:00.060
```

	SupplierCompanyName	TotalProducts	AverageUnitPrice
1	Supplier BWGYE	4	28.175
2	Supplier CIYNM	3	26.4833
3	Supplier ELCRN	3	18.0833
4	Supplier EQPNC	2	29.50
5	Supplier ERVYZ	2	15.725
6	Supplier FNUXM	2	11.125
7	Supplier GQRCV	5	35.57
8	Supplier JDNUG	2	14.025
9	Supplier JNNES	3	30.9333

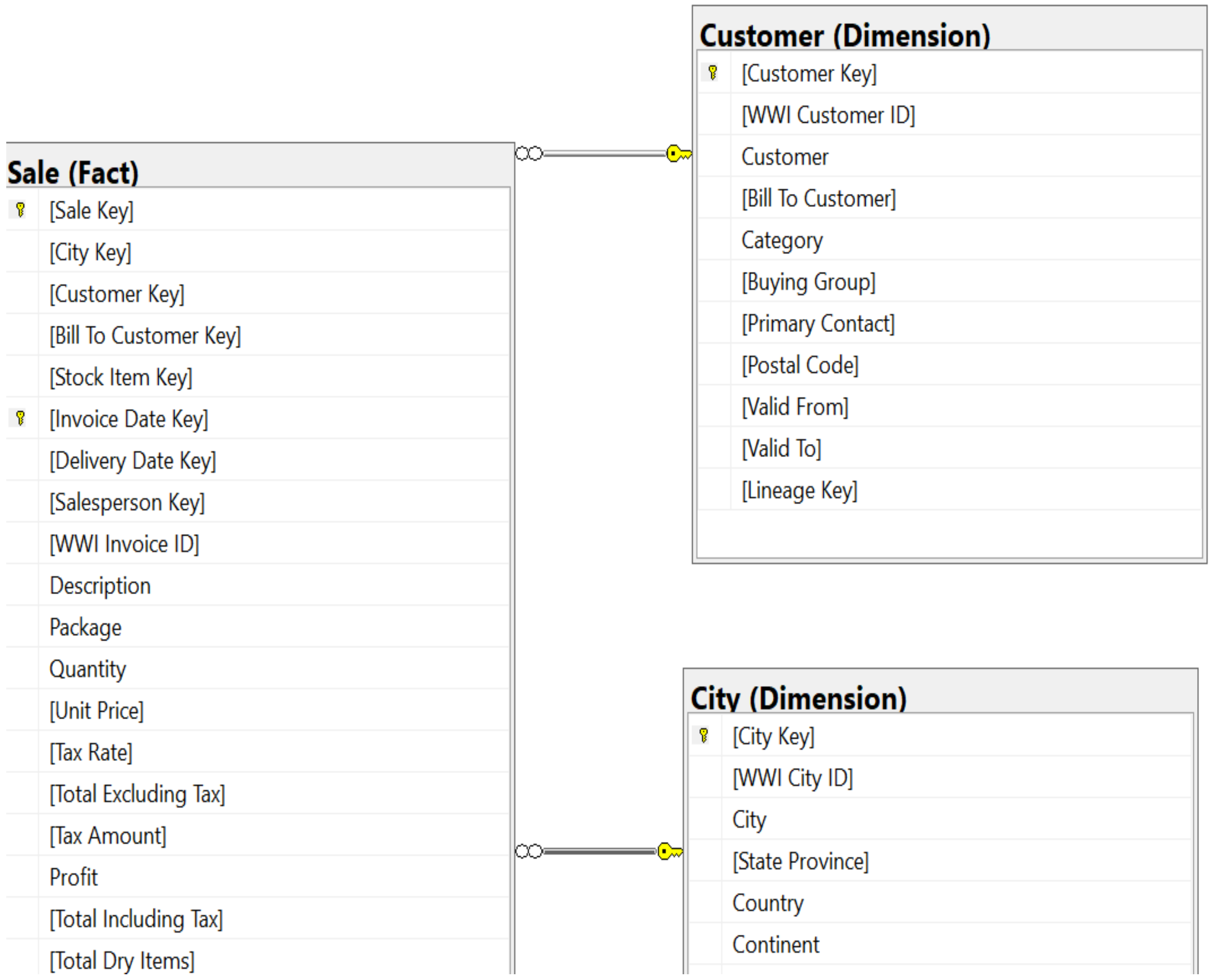
Sample JSON Output with total number of rows returned:



```
{
    "SupplierProductSummary": [
        {
            "SupplierCompanyName": "Supplier BWGYE",
            "TotalProducts": 4,
            "AverageUnitPrice": 28.1750
        },
        {
            "SupplierCompanyName": "Supplier CIYNM",
            "TotalProducts": 3,
            "AverageUnitPrice": 26.4833
        },
        {
            "SupplierCompanyName": "Supplier ELCRN",
            "TotalProducts": 3,
            "AverageUnitPrice": 18.0833
        },
        {
            "SupplierCompanyName": "Supplier EQPNC",
            "TotalProducts": 2,
            "AverageUnitPrice": 29.5000
        },
        {
            "SupplierCompanyName": "Supplier ERVYZ",
            "TotalProducts": 2,
            "AverageUnitPrice": 15.7250
        },
        {
            "SupplierCompanyName": "Supplier FNUXM",
            "TotalProducts": 2,
            "AverageUnitPrice": 11.1250
        },
```

MEDIUM QUERY

Use WideWorldImportersDW



PROPOSITION:

Formulate or adjust a query in the WideWorldImportersDW database to synthesize sales data per customer, incorporating the city context. This endeavor aims to present both the total and average sales amounts, uniquely identifying each customer by their key and including the city for further geographical insights.

Columns from their respective tables in the select clause:

Table Name	Column Name
Dimension.Customer	[Customer Key], Customer
Fact.Sale	[Total Including Tax], [Customer Key], [City Key]
Dimension.City	City, [City Key]

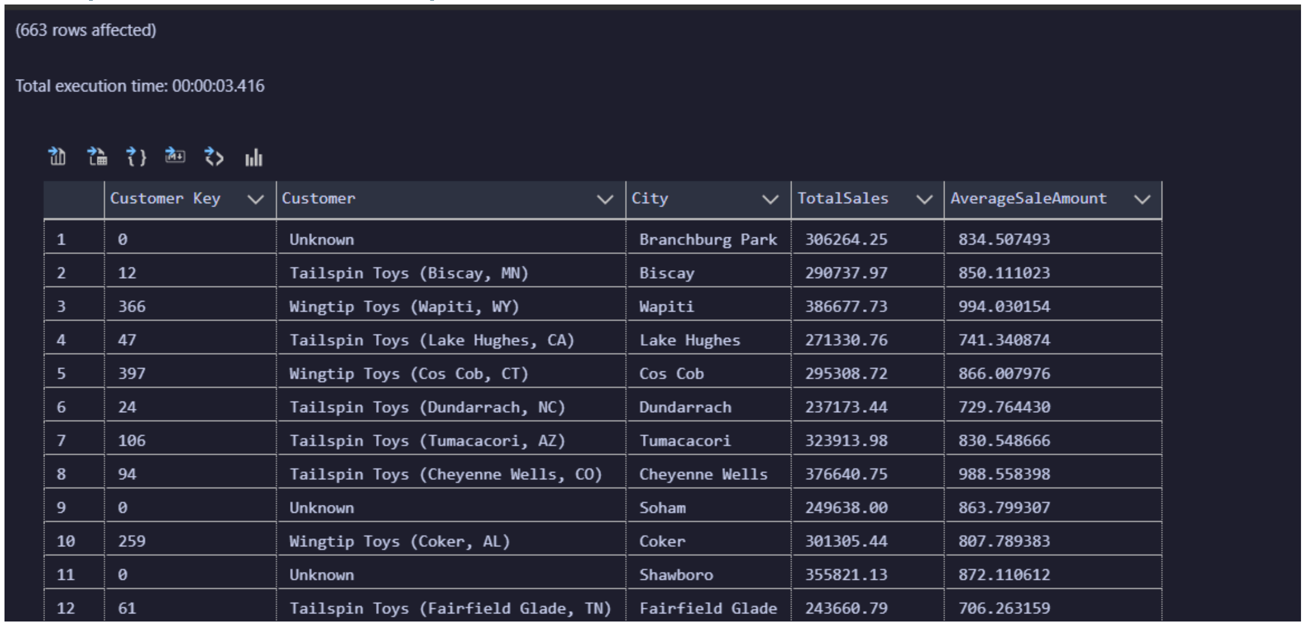
Order by :

Table Name	Column Name	Sort Order
Dimension.Customer	[Customer Key]	Asc
Dimension.Customer	City	Asc

Problem solving Query:

```
Use WideWorldImportersDW

;WITH SalesByCustomer AS (

    SELECT

        c.[Customer Key],

        c.Customer,

        ci.City,

        SUM(s.[Total Including Tax]) AS TotalSales,
        AVG(s.[Total Including Tax]) AS AverageSaleAmount

    FROM

        Dimension.Customer AS c

        JOIN Fact.Sale AS s ON c.[Customer Key] = s.[Customer Key]

        JOIN Dimension.City AS ci ON s.[City Key] = ci.[City Key]

    GROUP BY

        c.[Customer Key],

        c.Customer,

        ci.City

)

-- Relational output

SELECT [Customer Key], Customer, City, TotalSales, AverageSaleAmount

FROM SalesByCustomer


-- JSON output

FOR JSON PATH, ROOT('SalesByCustomer');
```

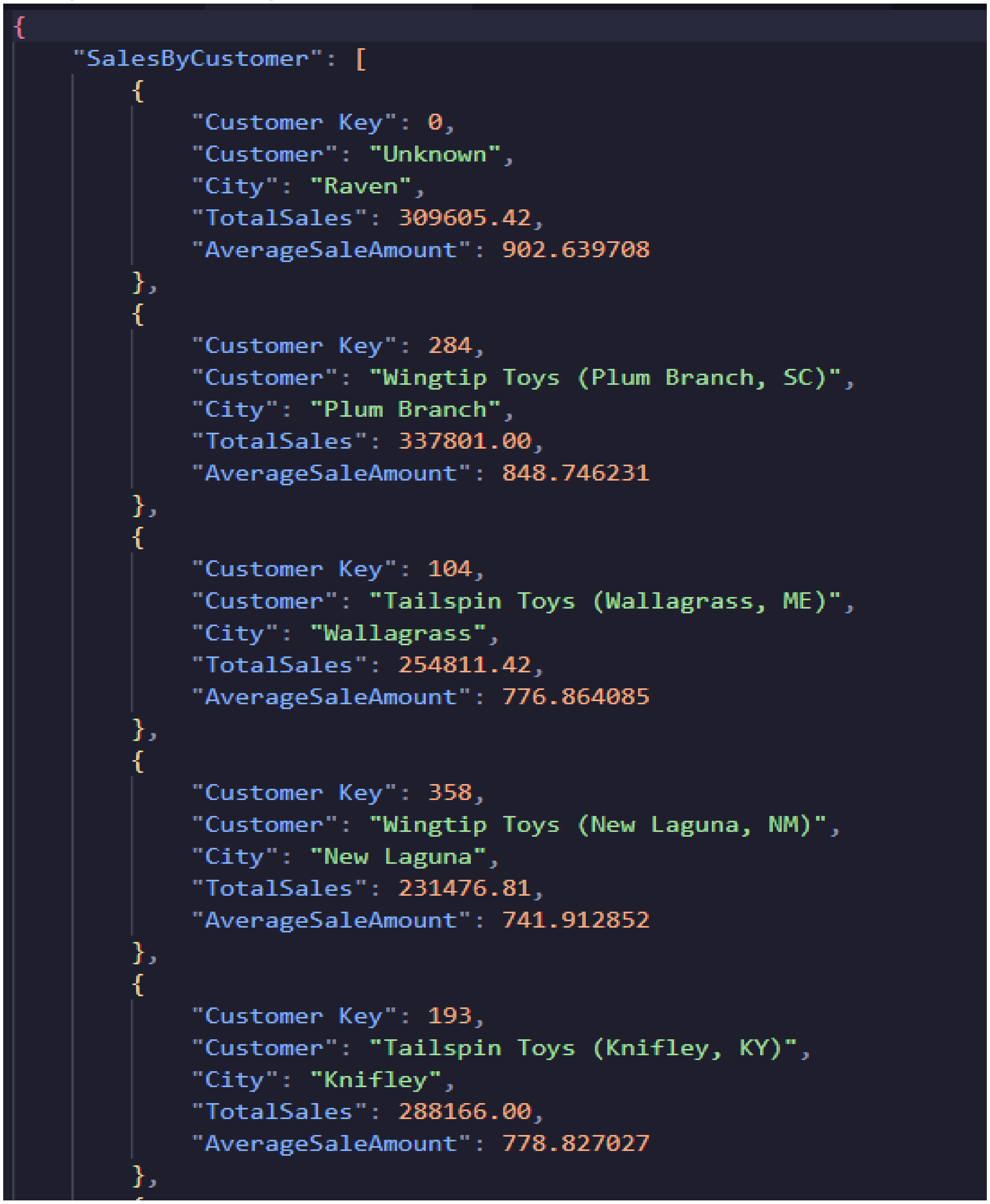
Sample Relational Output with total number of rows returned:



```
(663 rows affected)

Total execution time: 00:00:03.416
```

	Customer Key	Customer	City	TotalSales	AverageSaleAmount
1	0	Unknown	Branchburg Park	306264.25	834.507493
2	12	Tailspin Toys (Biscay, MN)	Biscay	290737.97	850.111023
3	366	Wingtip Toys (Wapiti, WY)	Wapiti	386677.73	994.030154
4	47	Tailspin Toys (Lake Hughes, CA)	Lake Hughes	271330.76	741.340874
5	397	Wingtip Toys (Cos Cob, CT)	Cos Cob	295308.72	866.007976
6	24	Tailspin Toys (Dundarrach, NC)	Dundarrach	237173.44	729.764430
7	106	Tailspin Toys (Tumacacori, AZ)	Tumacacori	323913.98	830.548666
8	94	Tailspin Toys (Cheyenne Wells, CO)	Cheyenne Wells	376640.75	988.558398
9	0	Unknown	Soham	249638.00	863.799307
10	259	Wingtip Toys (Coker, AL)	Coker	301305.44	807.789383
11	0	Unknown	Shawboro	355821.13	872.110612
12	61	Tailspin Toys (Fairfield Glade, TN)	Fairfield Glade	243660.79	706.263159

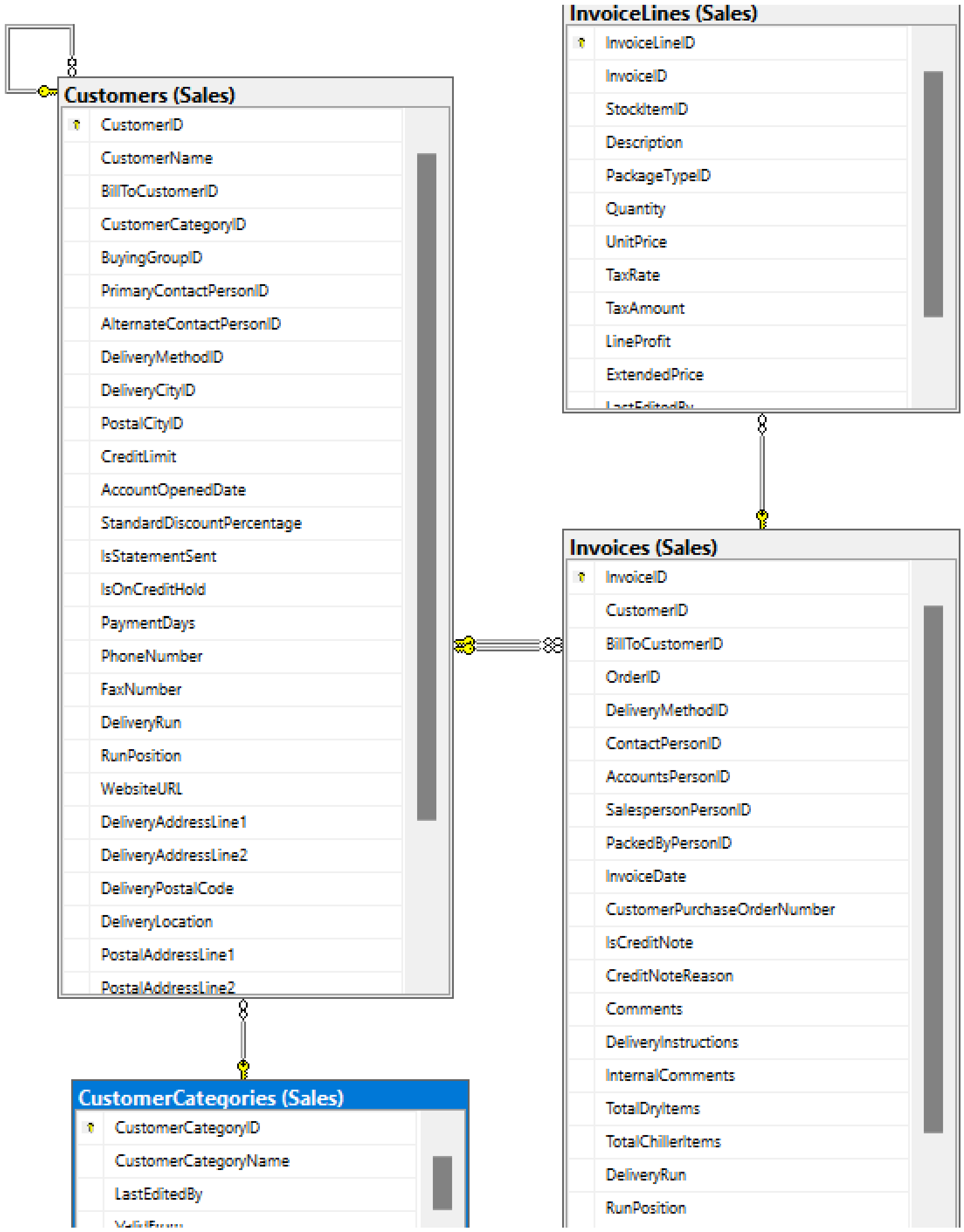
Sample JSON Output with total number of rows returned:



```
{
    "SalesByCustomer": [
        {
            "Customer Key": 0,
            "Customer": "Unknown",
            "City": "Raven",
            "TotalSales": 309605.42,
            "AverageSaleAmount": 902.639708
        },
        {
            "Customer Key": 284,
            "Customer": "Wingtip Toys (Plum Branch, SC)",
            "City": "Plum Branch",
            "TotalSales": 337801.00,
            "AverageSaleAmount": 848.746231
        },
        {
            "Customer Key": 104,
            "Customer": "Tailspin Toys (Wallagrass, ME)",
            "City": "Wallagrass",
            "TotalSales": 254811.42,
            "AverageSaleAmount": 776.864085
        },
        {
            "Customer Key": 358,
            "Customer": "Wingtip Toys (New Laguna, NM)",
            "City": "New Laguna",
            "TotalSales": 231476.81,
            "AverageSaleAmount": 741.912852
        },
        {
            "Customer Key": 193,
            "Customer": "Tailspin Toys (Knifley, KY)",
            "City": "Knifley",
            "TotalSales": 288166.00,
            "AverageSaleAmount": 778.827027
        },
```

MEDIUM QUERY

Use WorldWideImporters



PROPOSITION: Create or update a view named Sales.CustomerSalesSummary to store aggregated sales data for customer categories based on their purchases. Then, select and display CustomerCategoryName, TotalSales, and AverageSalePerInvoice for each customer category from the Sales.CustomerSalesSummary view.

Columns from their respective tables in the select clause:

Table Name	Column Name
Sales.Customers table	CustomerID CustomerCategoryID
Sales.Invoices table	CustomerID (for joining with Sales.Customers) InvoiceID InvoiceDate
Sales.InvoiceLines table	InvoiceID (for joining with Sales.Invoices) InvoiceLineID ExtendedPrice
Sales.CustomerCategories	CustomerCategoryID CustomerCategoryName

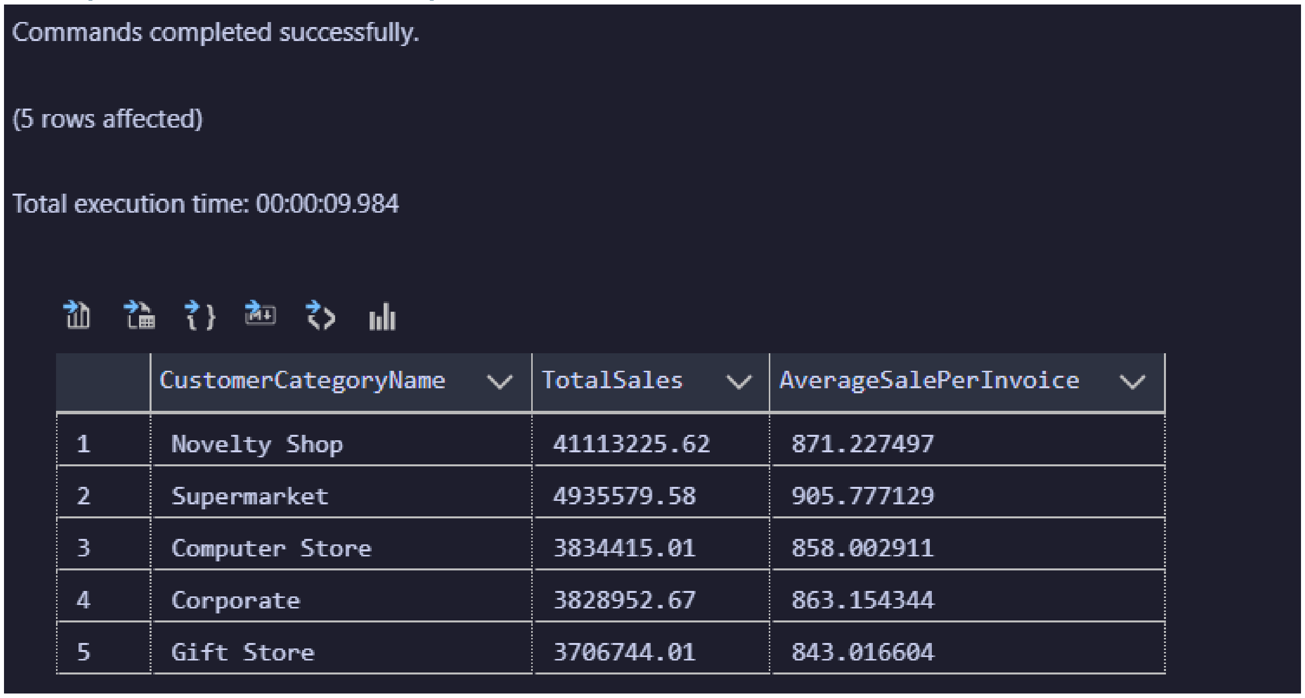
Order by :

Table Name	Column Name	Sort Order
Sales.InvoiceLines	TotalSales	Desc

Problem solving Query:

```
USE WideWorldimporters; -- Medium

GO


WITH CustomerSales AS (

    SELECT

        c.CustomerID,

        c.CustomerCategoryID,

        i.InvoiceDate,
        il.InvoiceLineID,

        il.ExtendedPrice

    FROM

        Sales.Customers c

    JOIN Sales.Invoices i ON c.CustomerID = i.CustomerID

    JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID

    WHERE

        i.InvoiceDate BETWEEN '2014-01-01' AND '2014-12-31'

),

AggregatedSales AS (

    SELECT

        CustomerCategoryID,

        SUM(ExtendedPrice) AS TotalSales,

        AVG(ExtendedPrice) AS AverageSalePerInvoice

    FROM

        CustomerSales

    GROUP BY

        CustomerCategoryID

)

SELECT

    cc.CustomerCategoryName,

    asales.TotalSales,

    asales.AverageSalePerInvoice

FROM

    AggregatedSales asales

JOIN Sales.CustomerCategories cc ON asales.CustomerCategoryID = cc.CustomerCategoryID

ORDER BY

    TotalSales DESC


-- JSON output

FOR JSON PATH, ROOT('CustomerSales');
```

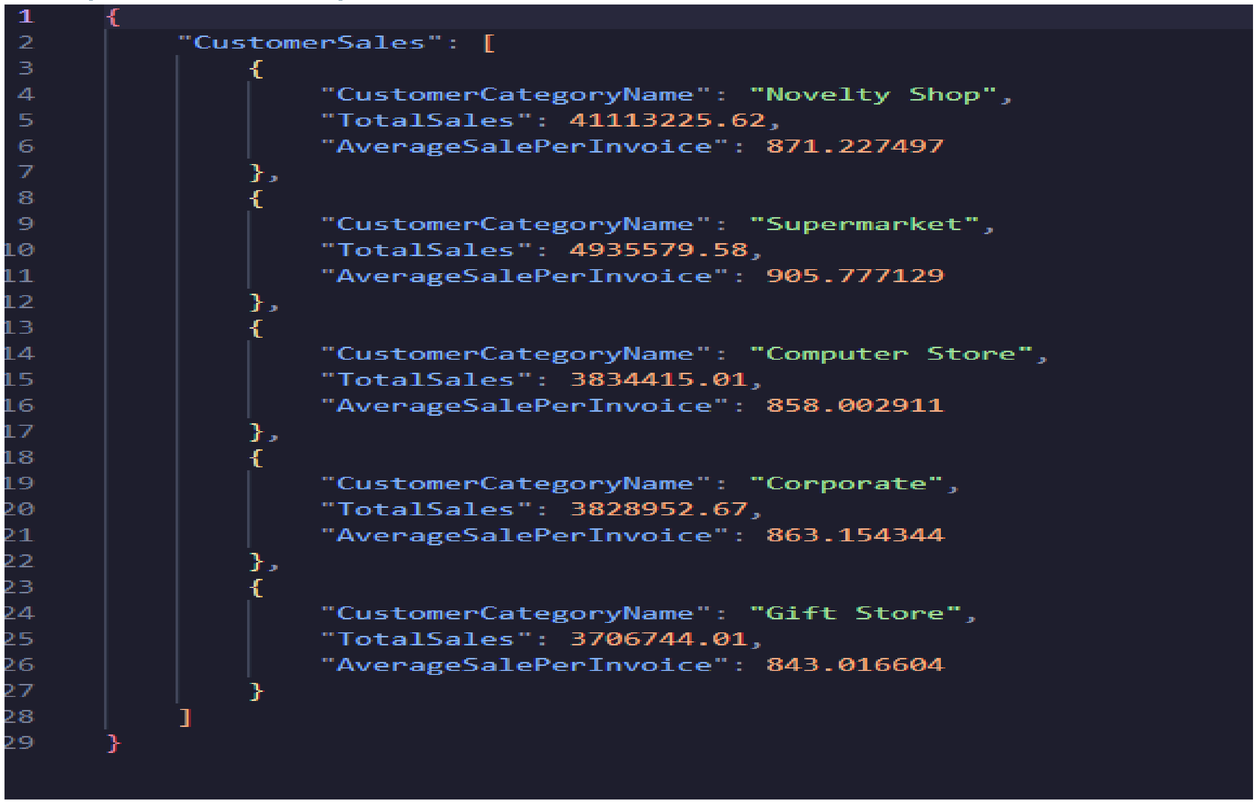
Sample Relational Output with total number of rows returned:



```
Commands completed successfully.

(5 rows affected)

Total execution time: 00:00:09.984
```

	CustomerCategoryName	TotalSales	AverageSalePerInvoice
1	Novelty Shop	41113225.62	871.227497
2	Supermarket	4935579.58	905.777129
3	Computer Store	3834415.01	858.002911
4	Corporate	3828952.67	863.154344
5	Gift Store	3706744.01	843.016604

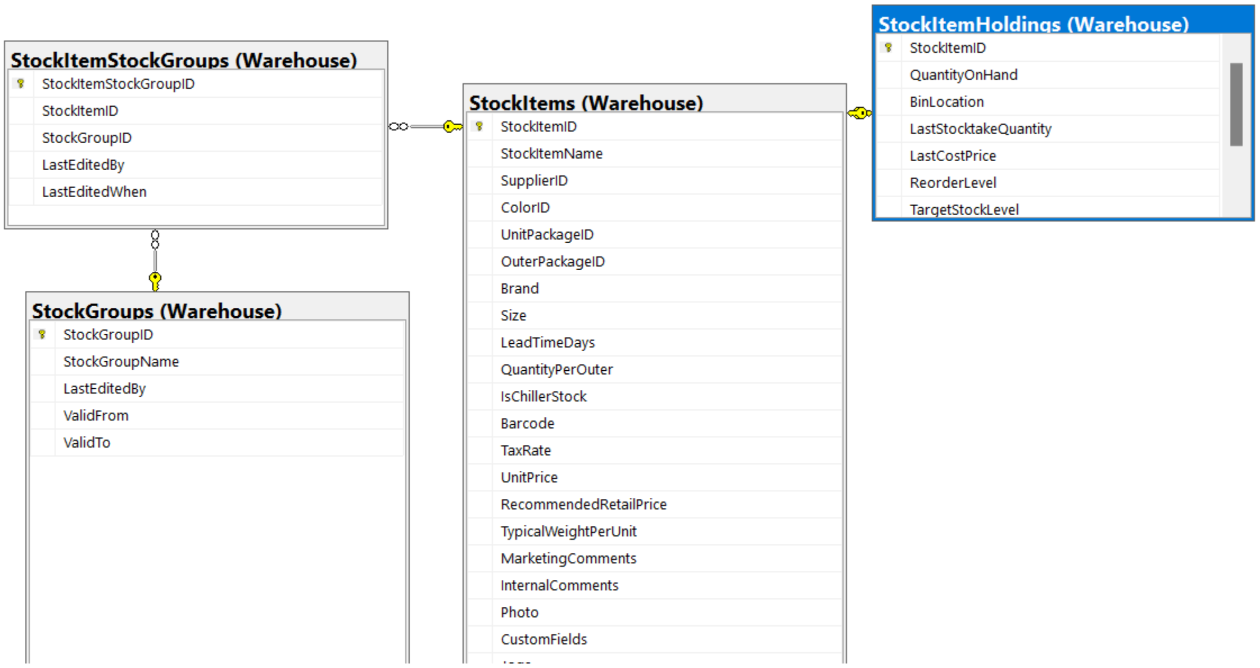
Sample JSON Output with total number of rows returned:



```
{
    "CustomerSales": [
        {
            "CustomerCategoryName": "Novelty Shop",
            "TotalSales": 41113225.62,
            "AverageSalePerInvoice": 871.227497
        },
        {
            "CustomerCategoryName": "Supermarket",
            "TotalSales": 4935579.58,
            "AverageSalePerInvoice": 905.777129
        },
        {
            "CustomerCategoryName": "Computer Store",
            "TotalSales": 3834415.01,
            "AverageSalePerInvoice": 858.002911
        },
        {
            "CustomerCategoryName": "Corporate",
            "TotalSales": 3828952.67,
            "AverageSalePerInvoice": 863.154344
        },
        {
            "CustomerCategoryName": "Gift Store",
            "TotalSales": 3706744.01,
            "AverageSalePerInvoice": 843.016604
        }
    ]
}
```

MEDIUM QUERIES

Use WideWorldimporters



Preposition: Create or update a view named Warehouse.StockSummary to store summarized information about stock items including their names, quantities on hand, and associated stock group names. Then, select and display StockItemName, QuantityOnHand, and StockGroupName for each stock item from the Warehouse.StockSummary view.

Columns from their respective tables in the select clause:

Table Name	Column Name
Warehouse.StockItemHoldings	StockItemID QuantityOnHand
Warehouse.StockItems	StockItemID StockItemName
Warehouse.StockItemStockGroups	StockItemID StockGroupID
Warehouse.StockGroups	StockGroupID StockGroupName

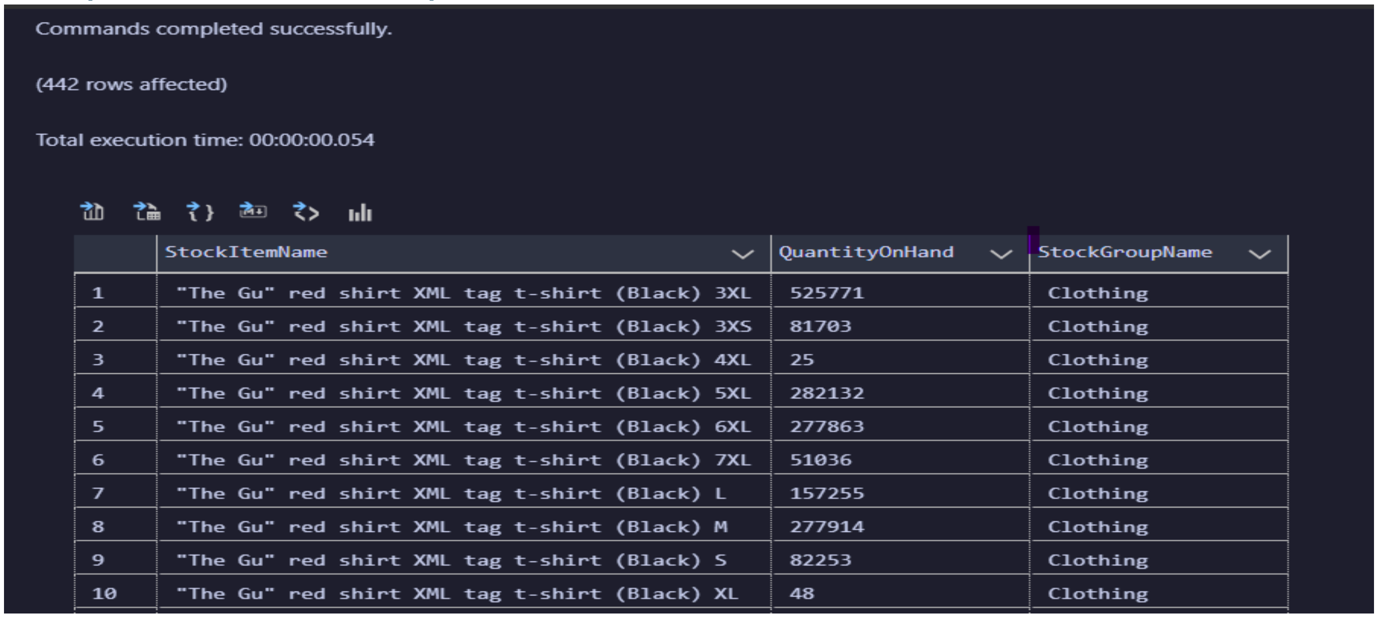
Order by :

Table Name	Column Name	Sort Order
Warehouse.StockGroups	StockGroupName	Ascending
Warehouse.StockItems	StockItemName	Ascending

Problem solving Query:

```
USE WideWorldimporters;

GO


WITH StockSummary AS (

    SELECT

        si.StockItemID,

        si.StockItemName,

        sish.QuantityOnHand,

        sig.StockGroupName

    FROM

        Warehouse.StockItemHoldings sish

    JOIN Warehouse.StockItems si ON sish.StockItemID = si.StockItemID

    JOIN Warehouse.StockItemStockGroups sisg ON si.StockItemID = sisg.StockItemID

    JOIN Warehouse.StockGroups sig ON sisg.StockGroupID = sig.StockGroupID

)

SELECT

    StockItemName,

    QuantityOnHand,

    StockGroupName

FROM

    StockSummary

ORDER BY

    StockGroupName, StockItemName


FOR JSON PATH, ROOT('StockSummary');
```

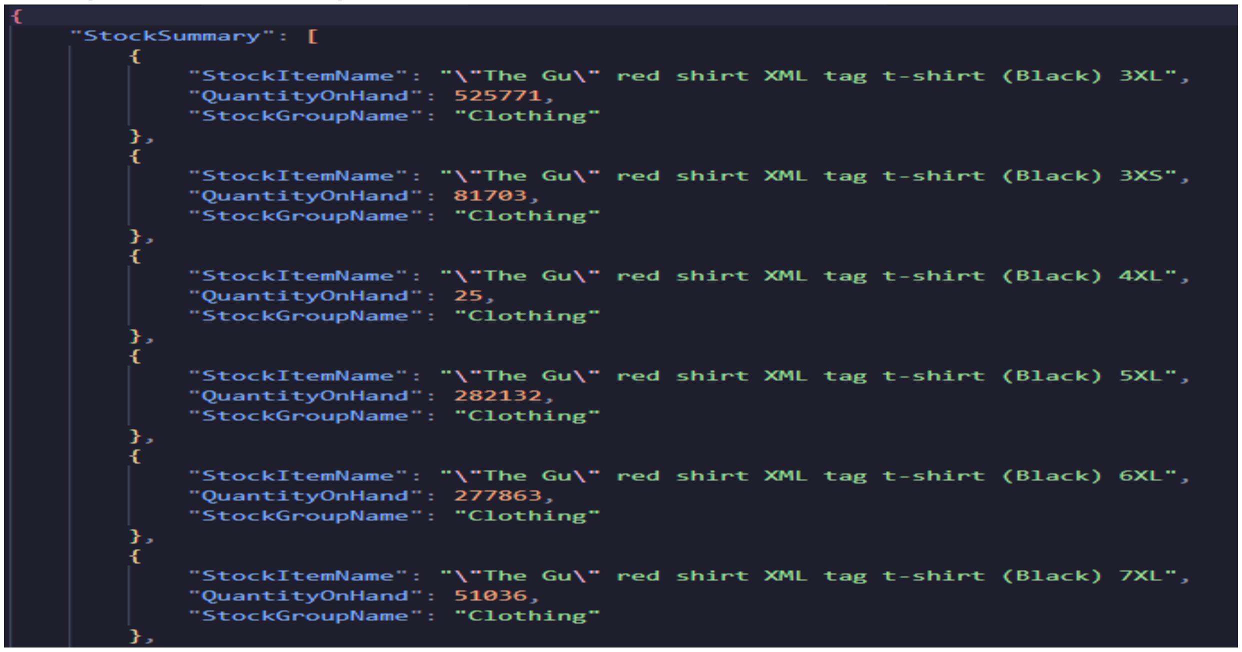
Sample Relational Output with total number of rows returned:



```
Commands completed successfully.

(442 rows affected)

Total execution time: 00:00:00.054
```

	StockItemName	QuantityOnHand	StockGroupName
1	"The Gu" red shirt XML tag t-shirt (Black) 3XL	525771	Clothing
2	"The Gu" red shirt XML tag t-shirt (Black) 3XS	81703	Clothing
3	"The Gu" red shirt XML tag t-shirt (Black) 4XL	25	Clothing
4	"The Gu" red shirt XML tag t-shirt (Black) 5XL	282132	Clothing
5	"The Gu" red shirt XML tag t-shirt (Black) 6XL	277863	Clothing
6	"The Gu" red shirt XML tag t-shirt (Black) 7XL	51036	Clothing
7	"The Gu" red shirt XML tag t-shirt (Black) L	157255	Clothing
8	"The Gu" red shirt XML tag t-shirt (Black) M	277914	Clothing
9	"The Gu" red shirt XML tag t-shirt (Black) S	82253	Clothing
10	"The Gu" red shirt XML tag t-shirt (Black) XL	48	Clothing

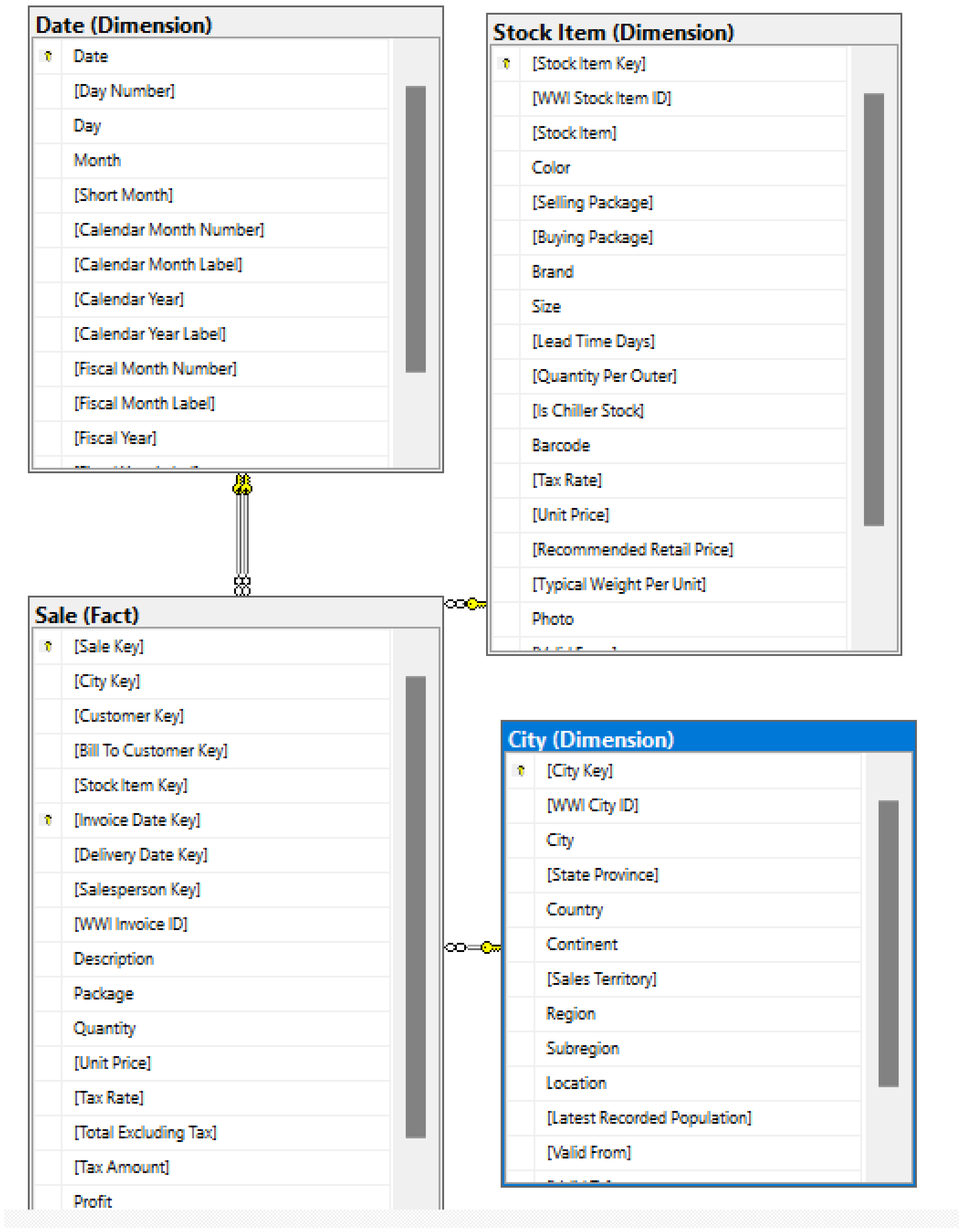
Sample JSON Output with total number of rows returned:



```
{
    "StockSummary": [
        {
            "StockItemName": "\"The Gu\" red shirt XML tag t-shirt (Black) 3XL",
            "QuantityOnHand": 525771,
            "StockGroupName": "Clothing"
        },
        {
            "StockItemName": "\"The Gu\" red shirt XML tag t-shirt (Black) 3XS",
            "QuantityOnHand": 81703,
            "StockGroupName": "Clothing"
        },
        {
            "StockItemName": "\"The Gu\" red shirt XML tag t-shirt (Black) 4XL",
            "QuantityOnHand": 25,
            "StockGroupName": "Clothing"
        },
        {
            "StockItemName": "\"The Gu\" red shirt XML tag t-shirt (Black) 5XL",
            "QuantityOnHand": 282132,
            "StockGroupName": "Clothing"
        },
        {
            "StockItemName": "\"The Gu\" red shirt XML tag t-shirt (Black) 6XL",
            "QuantityOnHand": 277863,
            "StockGroupName": "Clothing"
        },
        {
            "StockItemName": "\"The Gu\" red shirt XML tag t-shirt (Black) 7XL",
            "QuantityOnHand": 51036,
            "StockGroupName": "Clothing"
        },
```

COMPLEX QUERY

Use WideWorldimportersDW



Proposition: Create or update a view named Sales.RegionalTopSellingProducts to store summarized sales data by region and top-selling products for the year 2013. Then, select and display Region, StockItem, and TotalSales for the top 5 selling products in each region from the Sales.RegionalTopSellingProducts view.

Columns from their respective tables in the select clause:

Table Name	Column Name
Fact.Sale	[Total Including Tax] [Stock Item Key] [City Key] [Invoice Date Key]
Dimension.[Stock Item]	[Stock Item] (referred to as [Stock Item] in the query) [Stock Item Key]
Dimension.City	Region [City Key]
Dimension.Date	Date (used for joining with [Invoice Date Key]) [Calendar Year]

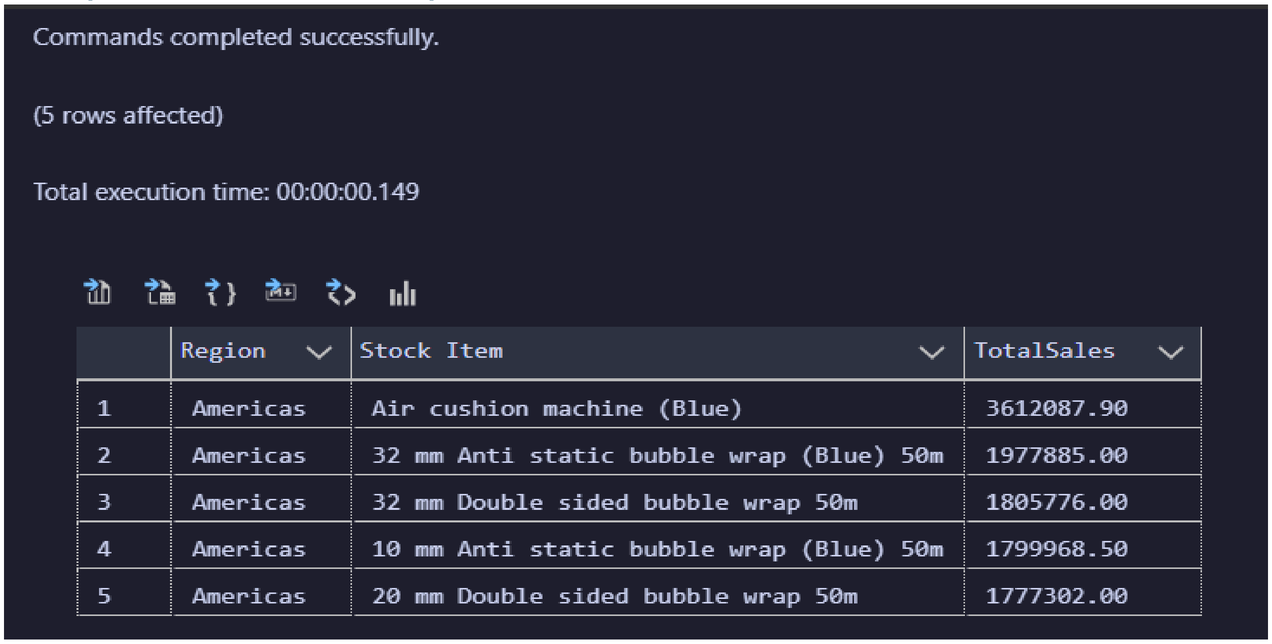
Order by :

Table Name	Column Name	Sort Order
Dimension.City	Region	Ascending
RankedRegionalSales CTE, TotalSales	SalesRank	Ascending

Problem solving Query:

```
USE WideWorldimportersDW; -- COMPLEX

GO


-- Define a CTE to summarize sales data by region and product category in 2020

WITH RegionalSalesData AS (

    SELECT

        ci.Region,

        si.[Stock Item],

        SUM(fs.[Total Including Tax]) AS TotalSales

    FROM

        Fact.Sale fs

    INNER JOIN Dimension.[Stock Item] si ON fs.[Stock Item Key] = si.[Stock Item Key]

    INNER JOIN Dimension.City ci ON fs.[City Key] = ci.[City Key]

    INNER JOIN Dimension.Date d ON fs.[Invoice Date Key] = d.Date
    WHERE
        d.[Calendar Year] = 2013
    GROUP BY
        ci.Region,
        si.[Stock Item]
),
-- Rank sales data within each region to identify top-selling products
RankedRegionalSales AS (
    SELECT
        Region,
        [Stock Item],
        TotalSales,
        RANK() OVER (PARTITION BY Region ORDER BY TotalSales DESC) AS SalesRank
    FROM
        RegionalSalesData
)
-- Select the top 5 selling products in each region
SELECT
    Region,
    [Stock Item],
    TotalSales
FROM
    RankedRegionalSales
WHERE
    SalesRank <= 5
ORDER BY
    Region,
    SalesRank

FOR JSON PATH, ROOT('RegionalSalesData');
```

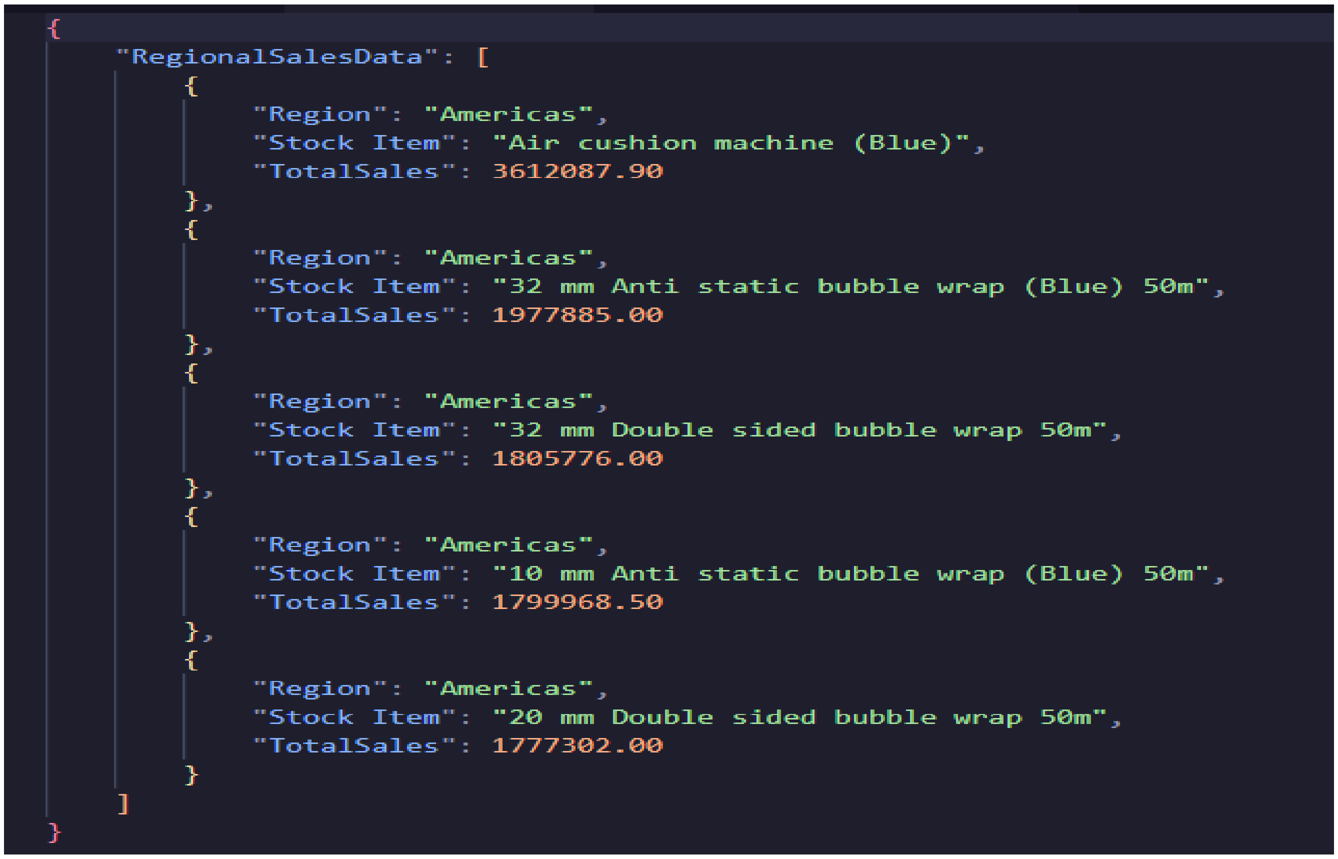
Sample Relational Output with total number of rows returned:



```
Commands completed successfully.

(5 rows affected)

Total execution time: 00:00:00.149
```

	Region	Stock Item	TotalSales
1	Americas	Air cushion machine (Blue)	3612087.90
2	Americas	32 mm Anti static bubble wrap (Blue) 50m	1977885.00
3	Americas	32 mm Double sided bubble wrap 50m	1805776.00
4	Americas	10 mm Anti static bubble wrap (Blue) 50m	1799968.50
5	Americas	20 mm Double sided bubble wrap 50m	1777302.00

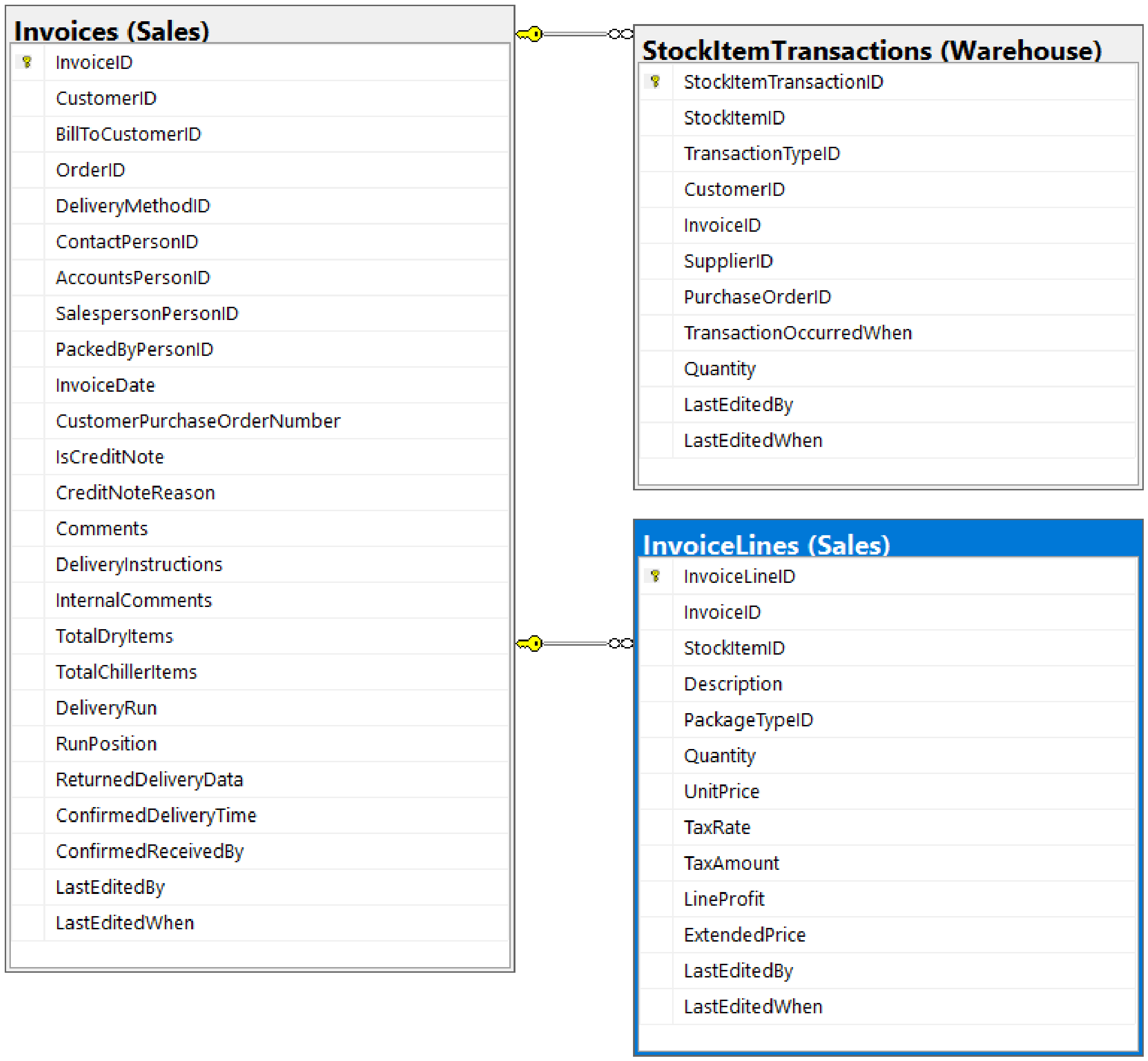
Sample JSON Output with total number of rows returned:



```
{
    "RegionalSalesData": [
        {
            "Region": "Americas",
            "Stock Item": "Air cushion machine (Blue)",
            "TotalSales": 3612087.90
        },
        {
            "Region": "Americas",
            "Stock Item": "32 mm Anti static bubble wrap (Blue) 50m",
            "TotalSales": 1977885.00
        },
        {
            "Region": "Americas",
            "Stock Item": "32 mm Double sided bubble wrap 50m",
            "TotalSales": 1805776.00
        },
        {
            "Region": "Americas",
            "Stock Item": "10 mm Anti static bubble wrap (Blue) 50m",
            "TotalSales": 1799968.50
        },
        {
            "Region": "Americas",
            "Stock Item": "20 mm Double sided bubble wrap 50m",
            "TotalSales": 1777302.00
        }
    ]
}
```

COMPLEX QUERY

Use WideWorldImporters



Proposition: Establish or revise a function called dbo.GetTotalInvoiceTax within the WideWorldImporters database. The function is designed to compute the total tax amount for a given invoice and return it as a decimal value with precision up to 18 digits and 2 decimal places. The function queries the Sales.InvoiceLines table, sums up the TaxAmount column values where the InvoiceID matches the provided @InvoiceID parameter, and stores the result in a variable named @TotalTax. This computed total tax amount is then returned by the function.

Columns from their respective tables in the select clause:

Table Name	Column Name
Sales.Invoices	TaxAmount InvoiceID InvoiceID CustomerID InvoiceDate
Sales.InvoiceLines	ExtendedPrice InvoiceLineID InvoiceID (used for joining with Sales.Invoices)
Warehouse.StockItemTransactions	StockItemID Quantity TransactionOccurredWhen InvoiceID

Order by :

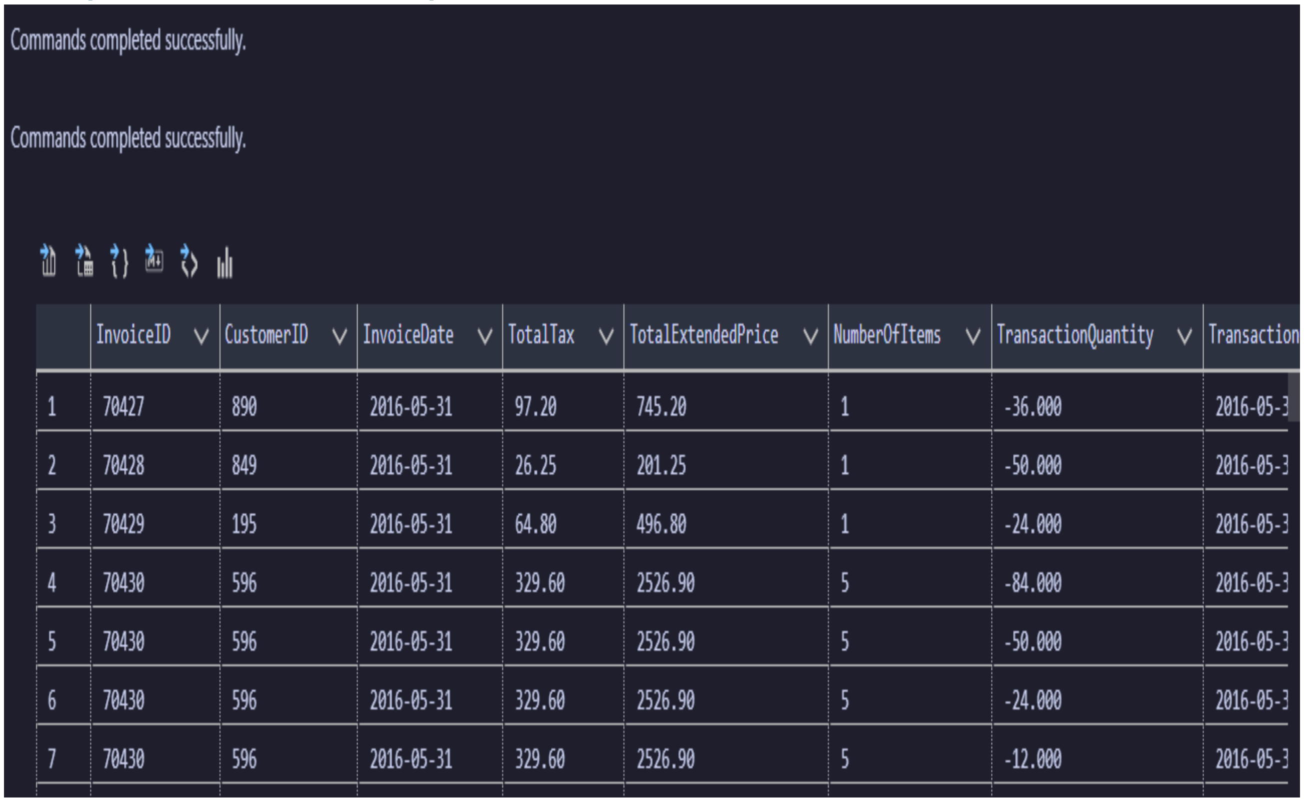
Table Name	Column Name	Sort Order
Sales.Invoices	InvoiceDate	DESC
Sales.Invoices	InvoiceID	ASC

```
USE WideWorldImporters --HARD

GO

CREATE OR ALTER FUNCTION  dbo.GetTotalInvoiceTax(@InvoiceID INT)

RETURNS DECIMAL(18,2)

AS

BEGIN

    DECLARE @TotalTax DECIMAL(18,2);

    SELECT @TotalTax = SUM(TaxAmount) FROM Sales.InvoiceLines WHERE InvoiceID =
@InvoiceID;

    RETURN @TotalTax;
```

```sql
END;
GO

;WITH InvoiceSummary AS (
    SELECT
        i.InvoiceID,
        i.CustomerID,
        i.InvoiceDate,
        dbo.GetTotalInvoiceTax(i.InvoiceID) AS TotalTax,
        SUM(il.ExtendedPrice) AS TotalExtendedPrice,
        COUNT(il.InvoiceLineID) AS NumberOfItems
    FROM Sales.Invoices i
    INNER JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID
    GROUP BY i.InvoiceID, i.CustomerID, i.InvoiceDate
),
StockTransactions AS (
    SELECT
        sit.StockItemID,
        sit.Quantity,
        sit.TransactionOccurredWhen,
        sit.InvoiceID
    FROM Warehouse.StockItemTransactions sit
    WHERE sit.InvoiceID IS NOT NULL
)
SELECT
    ISum.InvoiceID,
    ISum.CustomerID,
    ISum.InvoiceDate,
    ISum.TotalTax,
    ISum.TotalExtendedPrice,
    ISum.NumberOfItems,
    ST.Quantity AS TransactionQuantity,
    ST.TransactionOccurredWhen
```

```
FROM InvoiceSummary ISum

INNER JOIN StockTransactions ST ON ISum.InvoiceID = ST.InvoiceID

ORDER BY ISum.InvoiceDate DESC, ISum.InvoiceID


FOR JSON PATH, ROOT('dbo.GetTotalInvoiceTax');
```

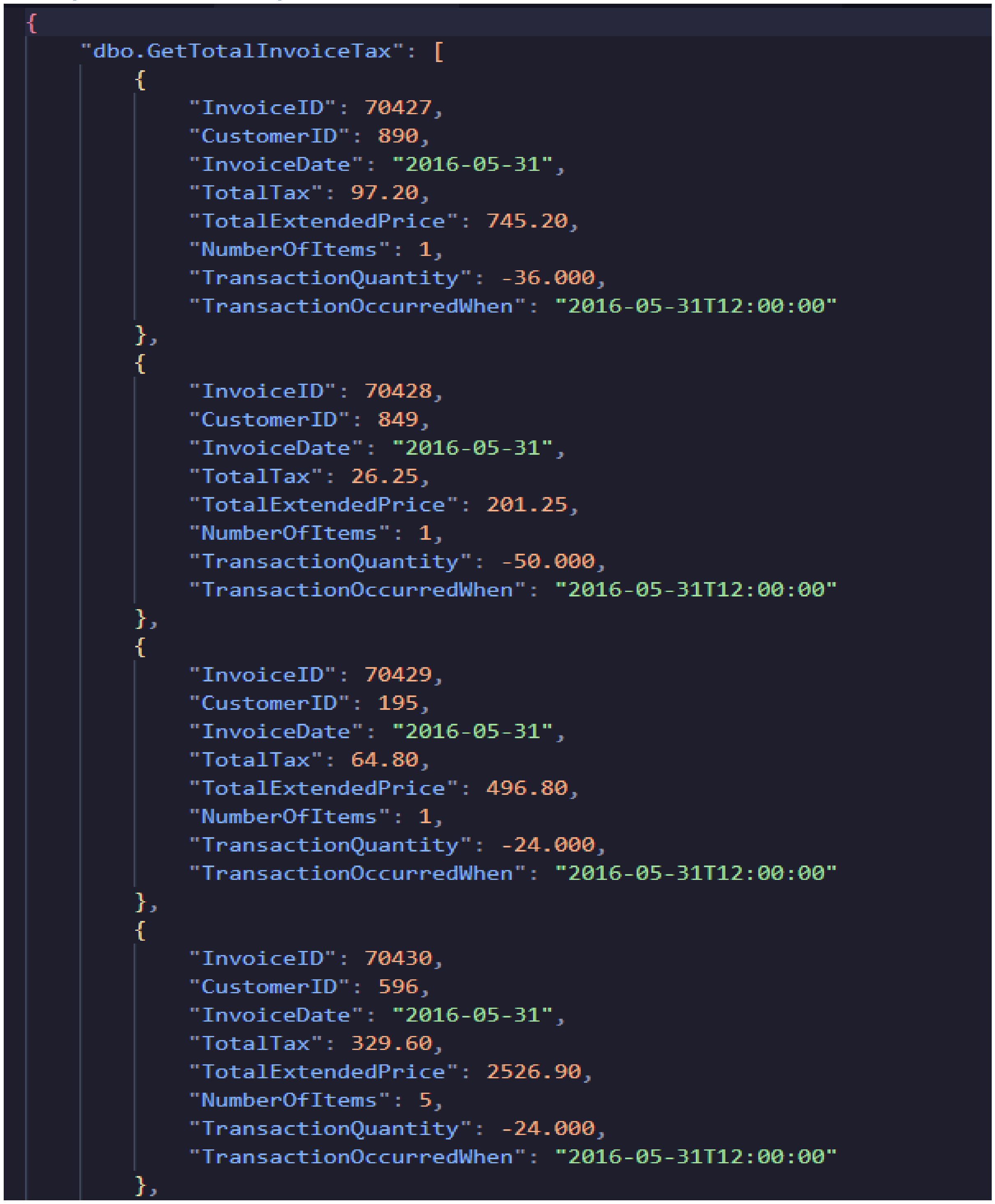
Sample Relational Output with total number of rows returned:



```
Commands completed successfully.

Commands completed successfully.
```

	InvoiceID	CustomerID	InvoiceDate	TotalTax	TotalExtendedPrice	NumberOfItems	TransactionQuantity	Transaction
1	70427	890	2016-05-31	97.20	745.20	1	-36.000	2016-05-3
2	70428	849	2016-05-31	26.25	201.25	1	-50.000	2016-05-3
3	70429	195	2016-05-31	64.80	496.80	1	-24.000	2016-05-3
4	70430	596	2016-05-31	329.60	2526.90	5	-84.000	2016-05-3
5	70430	596	2016-05-31	329.60	2526.90	5	-50.000	2016-05-3
6	70430	596	2016-05-31	329.60	2526.90	5	-24.000	2016-05-3
7	70430	596	2016-05-31	329.60	2526.90	5	-12.000	2016-05-3

Sample JSON Output with total number of rows returned:

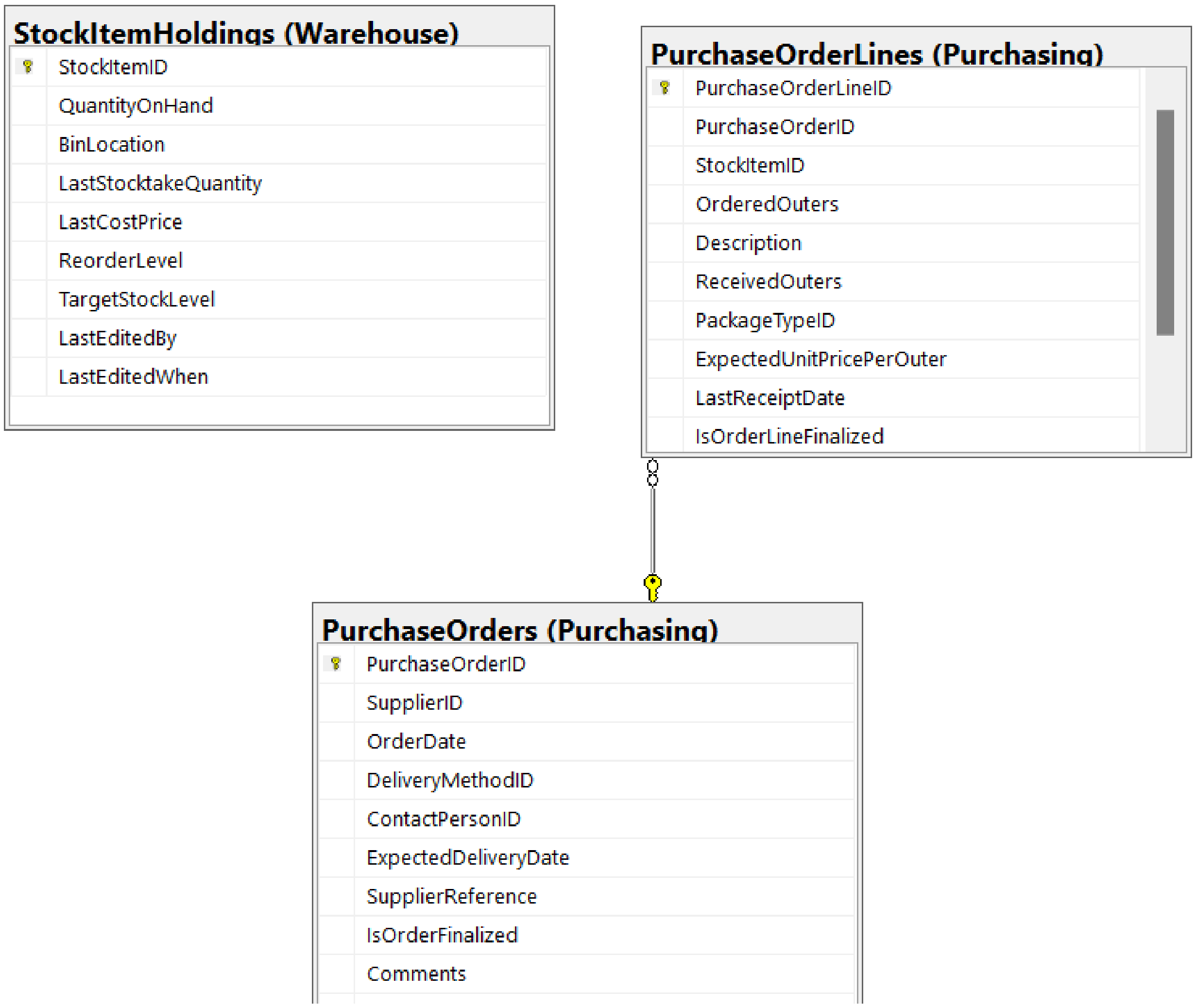


```
{
    "dbo.GetTotalInvoiceTax": [
        {
            "InvoiceID": 70427,
            "CustomerID": 890,
            "InvoiceDate": "2016-05-31",
            "TotalTax": 97.20,
            "TotalExtendedPrice": 745.20,
            "NumberOfItems": 1,
            "TransactionQuantity": -36.000,
            "TransactionOccurredWhen": "2016-05-31T12:00:00"
        },
        {
            "InvoiceID": 70428,
            "CustomerID": 849,
            "InvoiceDate": "2016-05-31",
            "TotalTax": 26.25,
            "TotalExtendedPrice": 201.25,
            "NumberOfItems": 1,
            "TransactionQuantity": -50.000,
            "TransactionOccurredWhen": "2016-05-31T12:00:00"
        },
        {
            "InvoiceID": 70429,
            "CustomerID": 195,
            "InvoiceDate": "2016-05-31",
            "TotalTax": 64.80,
            "TotalExtendedPrice": 496.80,
            "NumberOfItems": 1,
            "TransactionQuantity": -24.000,
            "TransactionOccurredWhen": "2016-05-31T12:00:00"
        },
        {
            "InvoiceID": 70430,
            "CustomerID": 596,
            "InvoiceDate": "2016-05-31",
            "TotalTax": 329.60,
            "TotalExtendedPrice": 2526.90,
            "NumberOfItems": 5,
            "TransactionQuantity": -24.000,
            "TransactionOccurredWhen": "2016-05-31T12:00:00"
        },
```

COMPLEX QUERY

USE WideWorldImporters

Columns from their respective tables in the select clause:



Proposition: Develop or update a function named dbo.AvgCostPerItem in the WideWorldImporters database. This function is designed to compute the average cost per item for a specified purchase order.

Columns from their respective tables in the select clause:

Table Name	Column Name
Purchasing.PurchaseOrderLines PurchaseOrderDetails CTE	PurchaseOrderID StockItemID OrderedOuters ReceivedOuters
Warehouse.StockItemHoldings PurchaseOrderDetails CTE	StockItemID QuantityOnHand LastCostPrice
Purchasing.PurchaseOrders	PurchaseOrderID SupplierID OrderDate

Order by :

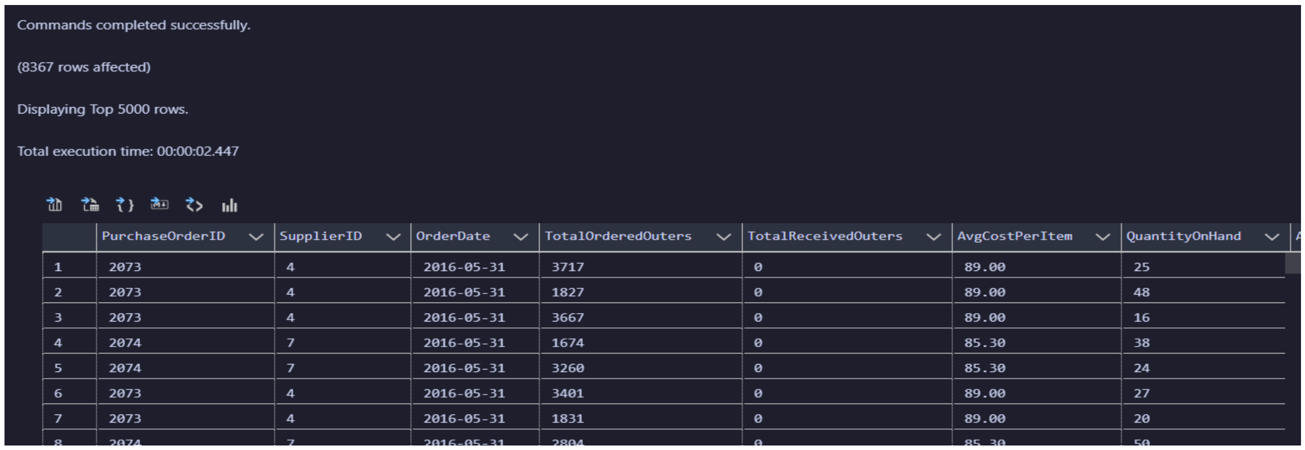
Table Name	Column Name	Sort Order
Purchasing.PurchaseOrders	OrderDate	DESC

Problem solving Query:

```
USE WideWorldImporters; --HARD

GO


CREATE OR ALTER FUNCTION dbo.AvgCostPerItem(@PurchaseOrderID INT)

RETURNS DECIMAL(18,2)

AS

BEGIN

    RETURN (

        SELECT AVG(ExpectedUnitPricePerOuter)

        FROM Purchasing.PurchaseOrderLines

        WHERE PurchaseOrderID = @PurchaseOrderID
    );
END;
GO

;WITH PurchaseOrderDetails AS (
    SELECT
        pol.PurchaseOrderID,
        pol.StockItemID,
        SUM(pol.OrderedOuters) AS TotalOrderedOuters,
        SUM(pol.ReceivedOuters) AS TotalReceivedOuters
    FROM Purchasing.PurchaseOrderLines pol
    GROUP BY pol.PurchaseOrderID, pol.StockItemID
), StockSummary AS (
    SELECT
        sih.StockItemID,
        sih.QuantityOnHand,
        AVG(sih.LastCostPrice) AS AvgLastCostPrice -- Assuming AVG is meaningful here
    FROM Warehouse.StockItemHoldings sih
    GROUP BY sih.StockItemID, sih.QuantityOnHand
)
SELECT
    po.PurchaseOrderID,
    po.SupplierID,
    po.OrderDate,
    POD.TotalOrderedOuters,
    POD.TotalReceivedOuters,
    dbo.AvgCostPerItem(po.PurchaseOrderID) AS AvgCostPerItem,
    SS.QuantityOnHand,
    SS.AvgLastCostPrice
FROM Purchasing.PurchaseOrders po
INNER JOIN PurchaseOrderDetails POD ON po.PurchaseOrderID = POD.PurchaseOrderID
LEFT JOIN StockSummary SS ON POD.StockItemID = SS.StockItemID
ORDER BY po.OrderDate DESC
```

```
FOR JSON PATH, ROOT('dbo.AvgCostPerItem');
```

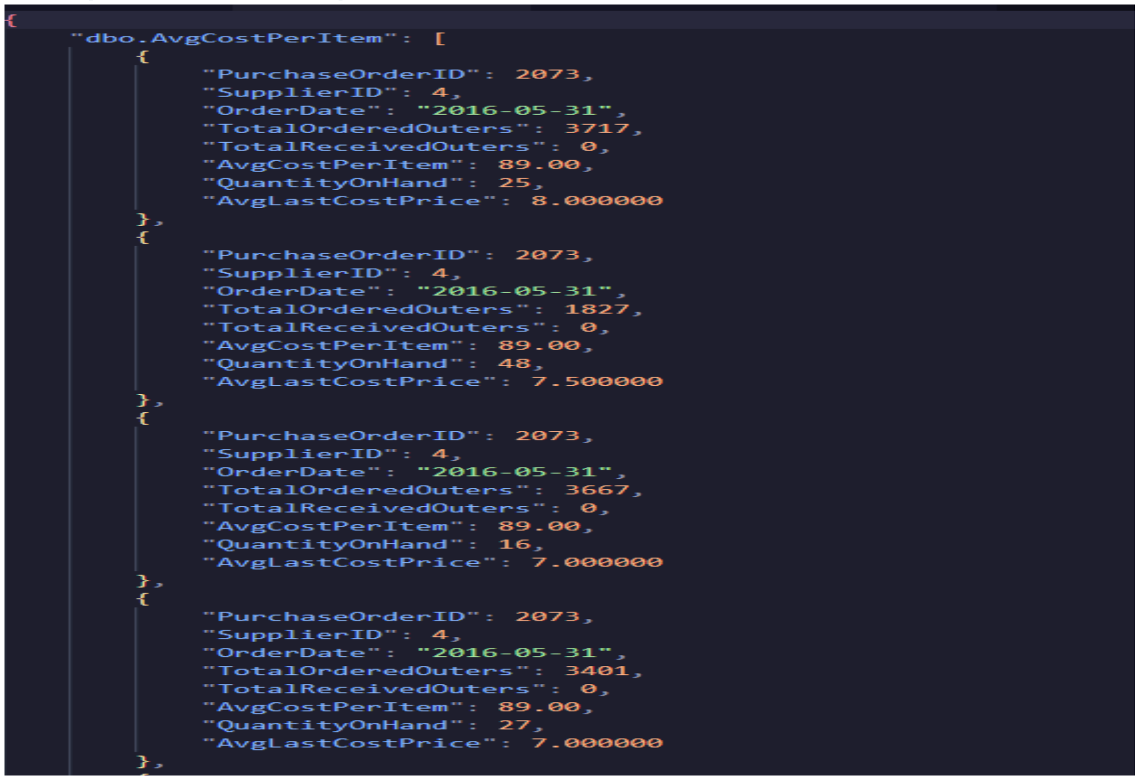
Sample Relational Output with total number of rows returned:



```
Commands completed successfully.

(8367 rows affected)

Displaying Top 5000 rows.

Total execution time: 00:00:02.447
```

	PurchaseOrderID	SupplierID	OrderDate	TotalOrderedOuters	TotalReceivedOuters	AvgCostPerItem	QuantityOnHand
1	2073	4	2016-05-31	3717	0	89.00	25
2	2073	4	2016-05-31	1827	0	89.00	48
3	2073	4	2016-05-31	3667	0	89.00	16
4	2074	7	2016-05-31	1674	0	85.30	38
5	2074	7	2016-05-31	3260	0	85.30	24
6	2073	4	2016-05-31	3401	0	89.00	27
7	2073	4	2016-05-31	1831	0	89.00	20
8	2074	7	2016-05-31	2804	0	85.30	50

Sample JSON Output with total number of rows returned:



```
{
    "dbo.AvgCostPerItem": [
        {
            "PurchaseOrderID": 2073,
            "SupplierID": 4,
            "OrderDate": "2016-05-31",
            "TotalOrderedOuters": 3717,
            "TotalReceivedOuters": 0,
            "AvgCostPerItem": 89.00,
            "QuantityOnHand": 25,
            "AvgLastCostPrice": 8.000000
        },
        {
            "PurchaseOrderID": 2073,
            "SupplierID": 4,
            "OrderDate": "2016-05-31",
            "TotalOrderedOuters": 1827,
            "TotalReceivedOuters": 0,
            "AvgCostPerItem": 89.00,
            "QuantityOnHand": 48,
            "AvgLastCostPrice": 7.500000
        },
        {
            "PurchaseOrderID": 2073,
            "SupplierID": 4,
            "OrderDate": "2016-05-31",
            "TotalOrderedOuters": 3667,
            "TotalReceivedOuters": 0,
            "AvgCostPerItem": 89.00,
            "QuantityOnHand": 16,
            "AvgLastCostPrice": 7.000000
        },
        {
            "PurchaseOrderID": 2073,
            "SupplierID": 4,
            "OrderDate": "2016-05-31",
            "TotalOrderedOuters": 3401,
            "TotalReceivedOuters": 0,
            "AvgCostPerItem": 89.00,
            "QuantityOnHand": 27,
            "AvgLastCostPrice": 7.000000
        },
```