Fall 2020

CS331 Project 1

Databases - Professor Heller - 10:45-12:00 TUES/THurs

Prepared by Harjit Liyal

Table of Contents

Preposition 1 - Simple Query 1 ………………………………………………………………………..Page 2

Preposition 2 - Simple Query 2 ………………………………………………………………………..Page 5

Preposition 3 - Simple Query 3 ………………………………………………………………………..Page 7

Preposition 4 - Simple Query 4…………………………………………………………………………Page 10

Preposition 5 - Simple Query 5…………………………………………………………………………Page 13

Preposition 6 - Medium Query 1………………………………………………………………………Page 16

Preposition 7 - Medium Query 2………………………………………………………………………Page 19

Preposition 8 - Medium Query 3………………………………………………………………………Page 22

Preposition 9 - Medium Query 4………………………………………………………………………Page 25

Preposition 10 - Medium Query 5…………………………………………………………………….Page 28

Preposition 11 - Medium Query 6…………………………………………………………………….Page 31

Preposition 12 - Medium Query 7…………………………………………………………………….Page 34

Preposition 13 - Medium Query 8…………………………………………………………………….Page 37

Preposition 14 - Complex Query 1…………………………………………………………………….Page 40

Preposition 15 - Complex Query 2…………………………………………………………………….Page 43

Preposition 16 - Complex Query 3…………………………………………………………………….Page 47

Preposition 17 - Complex Query 4…………………………………………………………………….Page 51

Preposition 18 - Complex Query 5…………………………………………………………………….Page 55

Preposition 19 - Complex Query 6…………………………………………………………………....Page 59

Preposition 20 - Complex Query 7…………………………………………………………………....Page 63

All prepositions follow the format:

1. Preposition explaining what the query returns using given database and tables
2. Standard and key view of the tables used
3. Chart that states the table name, columns, and the order by
4. Example relational code solution with screenshot of results and number of rows returned
5. Example JSON code solution with screenshot of results and number of objects returned

Note: Follow the page numbers on the bottom of the document when navigating using Table of Contents.

**Simple Query 1**

Return all customer ids, their country, and the date they ordered

Use TSQLV4 Database, Sales.Customer, Sales.Orders tables

**Standard view**



**Key view**



**Columns from tables**

|  |  |
| --- | --- |
| Table Name | Column Names |
| Sales.Customers | Custid  country |
| Sales.Orders | orderdate |

**Order By**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Name | Sort Order |
| Sales.Customers | custid | ASC |
| Sales.Customers | country | ASC |
| Sales.Orders | orderdate | ASC |

**Sample relational Solution Query Without JSON**

USE TSQLV4;

SELECT E.custid,

E.country,

F.orderdate

FROM Sales.Customers AS E

INNER JOIN Sales.Orders AS F

ON F.custid = E.custid

ORDER BY E.custid

--FOR JSON PATH, ROOT ('custid'), INCLUDE\_NULL\_VALUES;

**Solution Query With JSON**

USE TSQLV4;

SELECT E.custid,

E.country,

F.orderdate

FROM Sales.Customers AS E

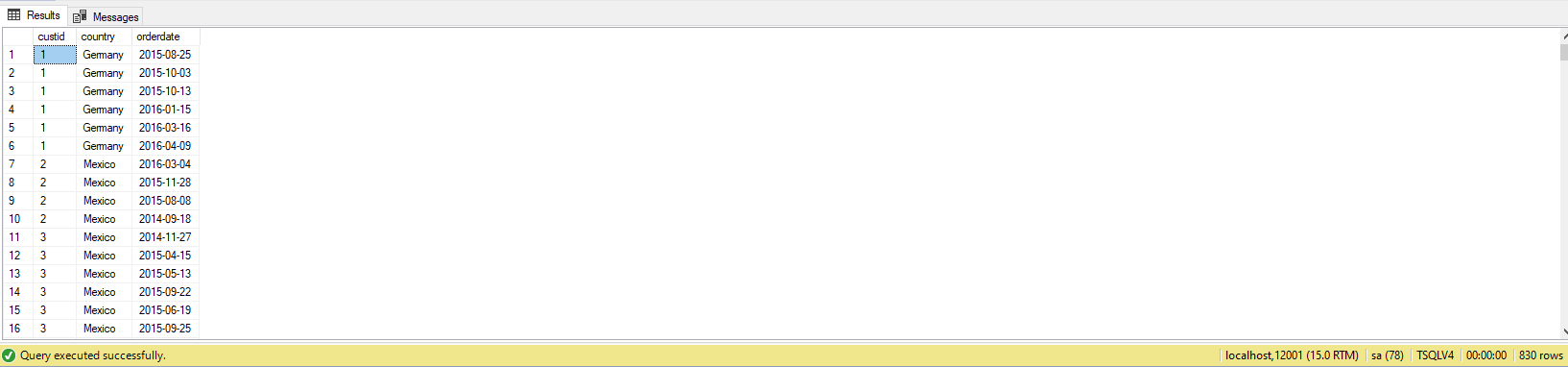
INNER JOIN Sales.Orders AS F

ON F.custid = E.custid

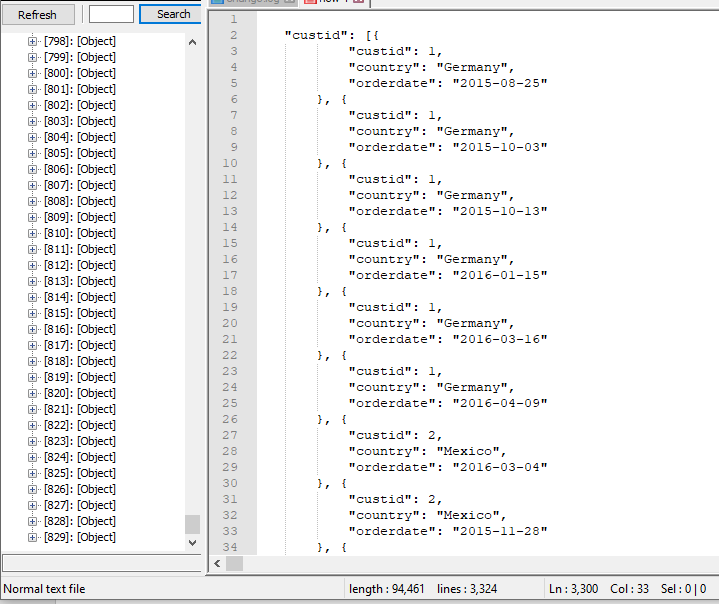
ORDER BY E.custid

FOR JSON PATH, ROOT('custid'), INCLUDE\_NULL\_VALUES;

**Sample Relational Output (830 Rows returned)**



**Sample JSON Output (830 Objects returned)**



**Simple Query 2**

Returns a inner join of all product ids mapped to order id 42

Use Northwinds2020TSQLV6 Database and Production.Product and Sales.OrderDetail tables

**Standard View**

****

**Key View**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order by |
| Production.Product | productid | ASC |
| Sales.OrderDetail | orderid | ASC |

**Solution Query Without JSON**

USE Northwinds2020TSQLV6;

SELECT E.ProductId,

F.OrderId

FROM Production.[Product] AS E

INNER JOIN Sales.[OrderDetail] AS F

ON F.ProductId = E.ProductId

WHERE E.ProductId = 42;

--FOR JSON PATH, ROOT ('custid'), INCLUDE\_NULL\_VALUES;

**Solution Query With JSON**

USE Northwinds2020TSQLV6;

SELECT E.ProductId,

F.OrderId

FROM Production.[Product] AS E

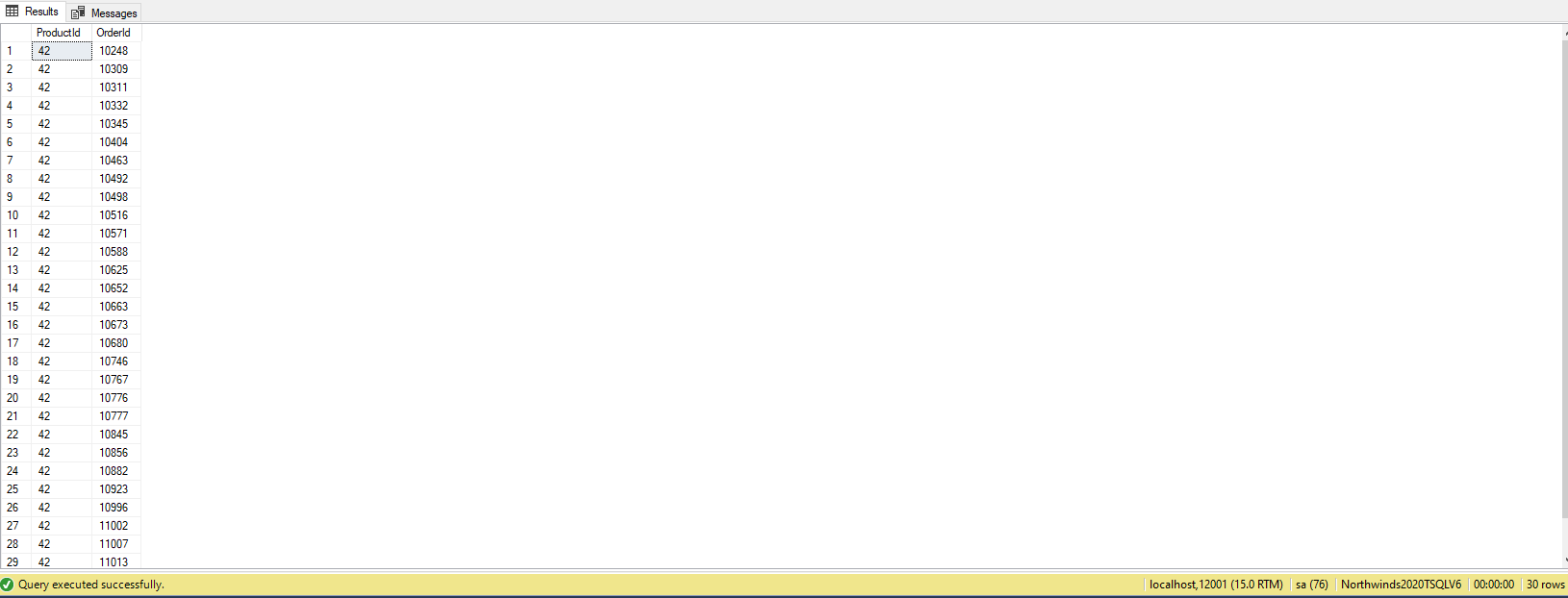
INNER JOIN Sales.[OrderDetail] AS F

ON F.ProductId = E.ProductId

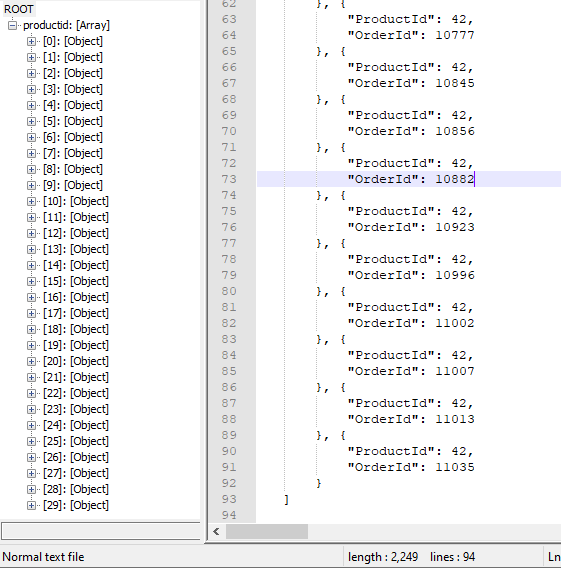
WHERE E.ProductId = 42

FOR JSON PATH, ROOT ('productid'), INCLUDE\_NULL\_VALUES;

**Sample Relational Output (30 Rows returned)**



**Sample JSON Output (30 Objects returned)**



**Simple Query 3**

Returns all Customer keys and Employee keys with their first and last name with the city they are from

Use AdventureWorksDW2017 Database and dbo.DimCustomer and dbo.DimGeography tables

**Standard table view**

****

**Key view**

****

**Table Names with column and order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Dbo.Geography | CustomerKey  FirstName  LastName | ASC  ASC  ASC |
| Dbo.DimCustomer | City | ASC |

**Sample relational solution and output (18484 rows returned)**

USE AdventureWorksDW2017;

SELECT E.CustomerKey,

E.FirstName,

E.LastName,

F.EmployeeKey,

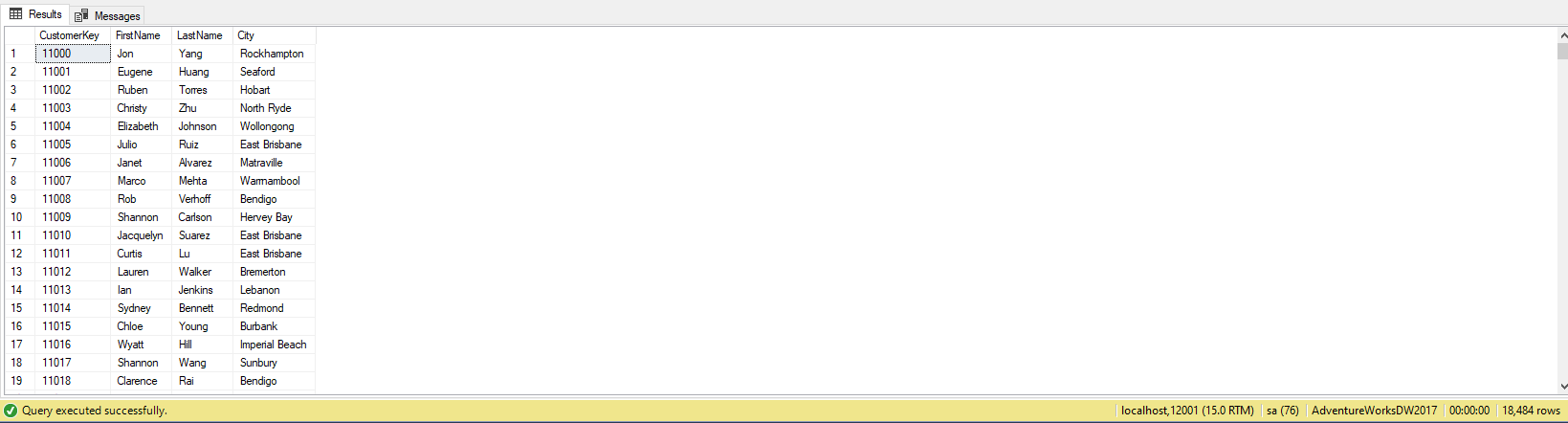
F.FirstName,

F.LastName

FROM dbo.[DimCustomer] AS E

FULL OUTER JOIN dbo.[DimEmployee] AS F

ON E.CustomerKey = F.EmployeeKey;



**Sample JSON solution and output (18780 Objects returned)**

USE AdventureWorksDW2017;

SELECT E.CustomerKey,

E.FirstName,

E.LastName,

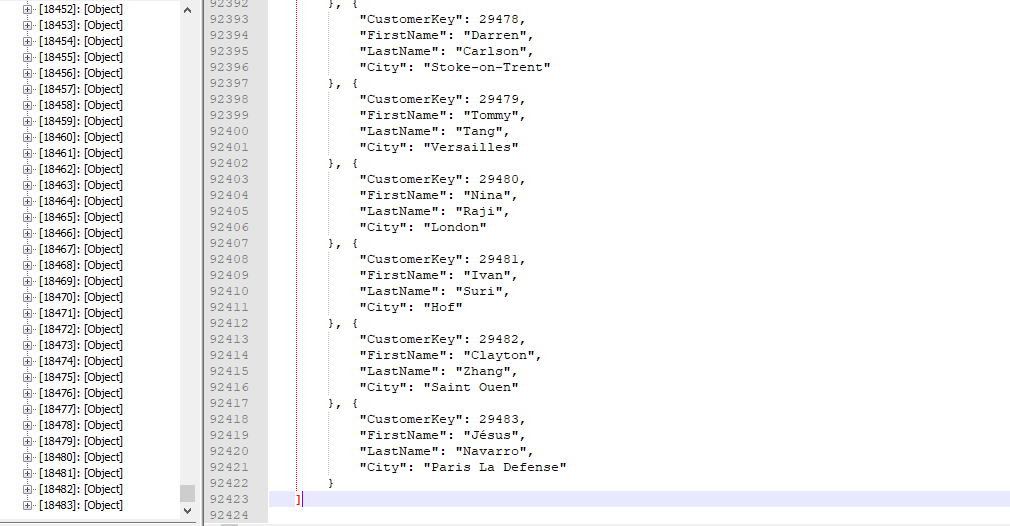
F.City

FROM dbo.[DimCustomer] AS E

INNER JOIN dbo.[DimGeography] AS F

ON F.GeographyKey = E.GeographyKey

FOR JSON PATH, ROOT('E.CustomerKey'), INCLUDE\_NULL\_VALUES;



**Simple Query 4**

Performs a left join and returns the purchase order id with the amount of units the customer ordered where the quantity is greater than 10 and not null

Use AdventureWorksDw database and PurchaseOrderDetail and Sales.Customer Tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| PurchaseOrderDetail | PurchaseOrderID  OrderQty | ASC  ASC |
| Sales.Customer | CustomerID | ASC |

**Sample relational solution and output (809 rows returned)**

USE AdventureWorks2017;

SELECT E.PurchaseOrderID,

E.OrderQty,

F.CustomerID

FROM Purchasing.[PurchaseOrderDetail] AS E

LEFT JOIN Sales.[Customer] AS F

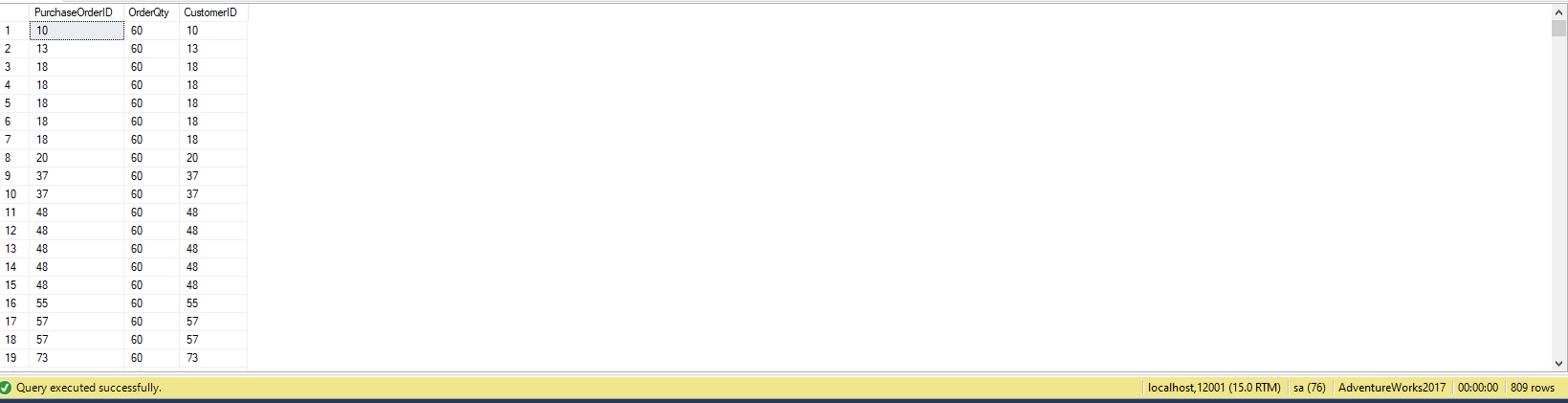
ON E.PurchaseOrderID = F.CustomerID

WHERE E.OrderQty > 10

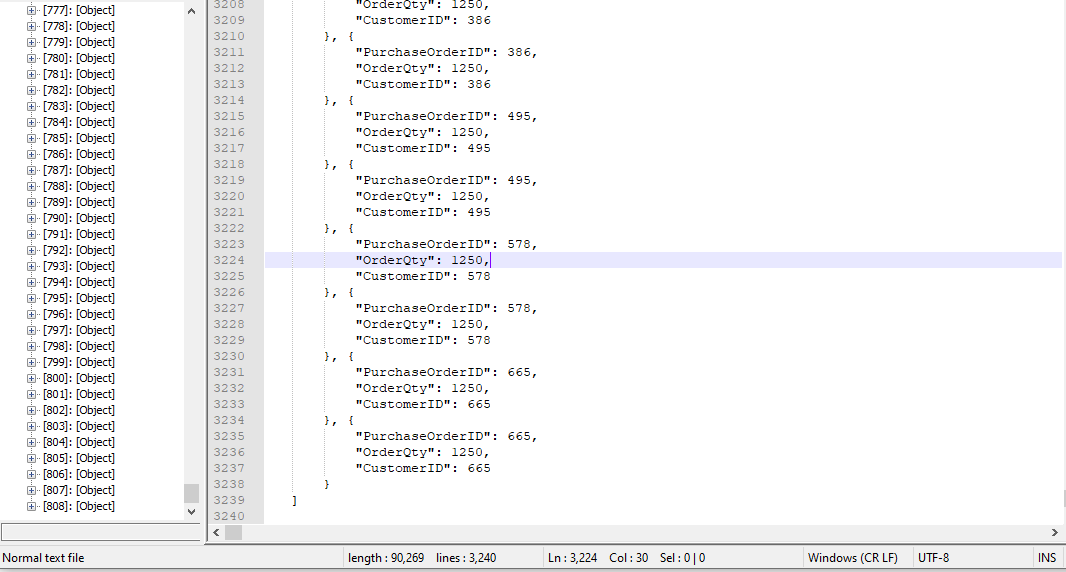
AND F.CustomerID IS NOT NULL

ORDER BY E.OrderQty

FOR JSON PATH, ROOT('E.CustomerKey'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution and output (809 objects returned)**



**Simple Query 5**

Performs a full outer join and returns the employee id and employee name with the order id and date they handled the order

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| HR.Employees | Empid  Fullname | ASC  ASC |
| Sales.Orders | OrderId  OrderDate | ASC  ASC |

**Sample Relation Solution with output (830 rows returned)**

USE TSQLV4;

SELECT E.empid AS EmployeeID,

E.FullName AS EmployeeName,

F.orderid,

F.orderdate

FROM HR.Employees AS E

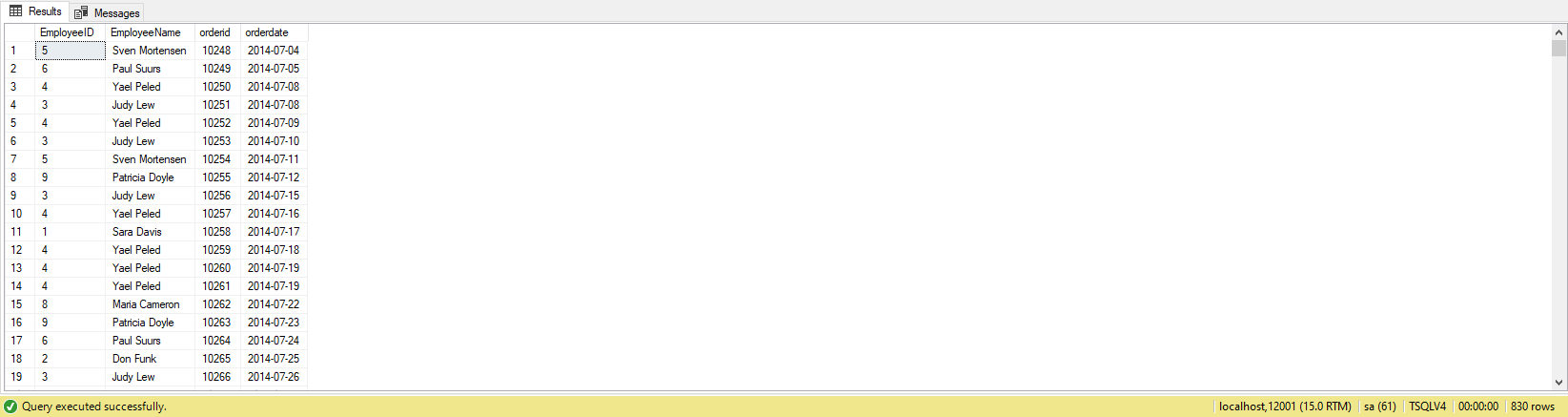
FULL OUTER JOIN Sales.Orders AS F

ON F.empid = E.empid

WHERE E.empid IS NOT NULL

AND F.orderid IS NOT NULL;

--FOR JSON PATH, ROOT('E.CustomerKey'), INCLUDE\_NULL\_VALUES;



**Sample JSON Solution with output (830 Objects returned)**

USE TSQLV4;

SELECT E.empid AS EmployeeID,

E.FullName AS EmployeeName,

F.orderid,

F.orderdate

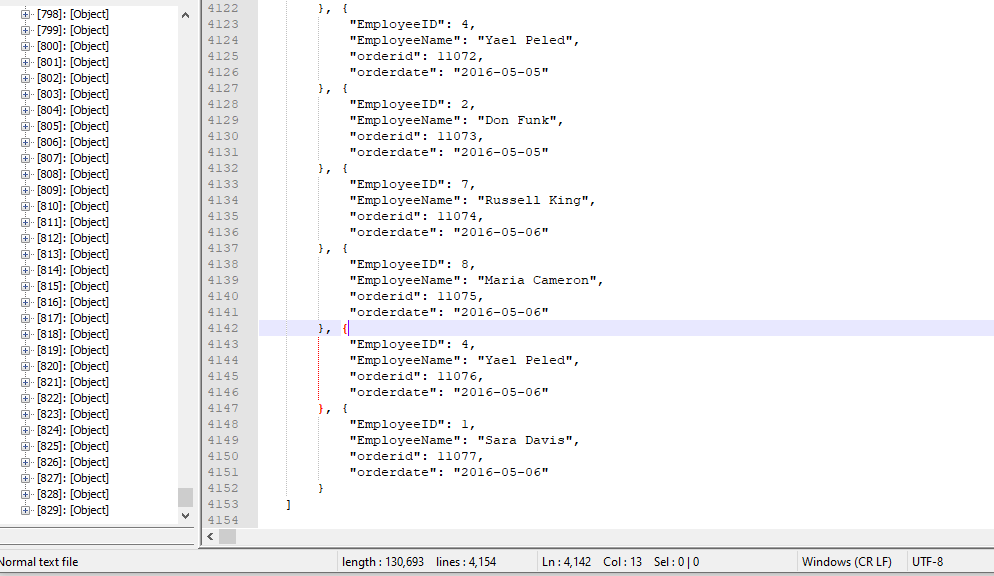
FROM HR.Employees AS E

FULL OUTER JOIN Sales.Orders AS F

ON F.empid = E.empid

WHERE E.empid IS NOT NULL

AND F.orderid IS NOT NULL

FOR JSON PATH, ROOT('E.CustomerKey'), INCLUDE\_NULL\_VALUES;

**Medium Query 1**

Performs an inner join on the two tables and returns all order id, unit price, order id and order date in 2016 in descending order

Use TSQLV4 database and Sales.Orders and Sales.OrderDetails tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Dbo.OrderDetails | OrderId  Unitprice | ASC  ASC |
| Dbo.Orders | OrderDate | DESC |

**Sample relation solution with output (686 rows returned)**

USE TSQLV4;

SELECT O.orderid,

O.unitprice,

P.orderdate

FROM dbo.Orderdetails AS O

INNER JOIN dbo.Orders AS P

ON P.orderid = O.orderid

AND YEAR(P.orderdate) = 2016

GROUP BY O.orderid,

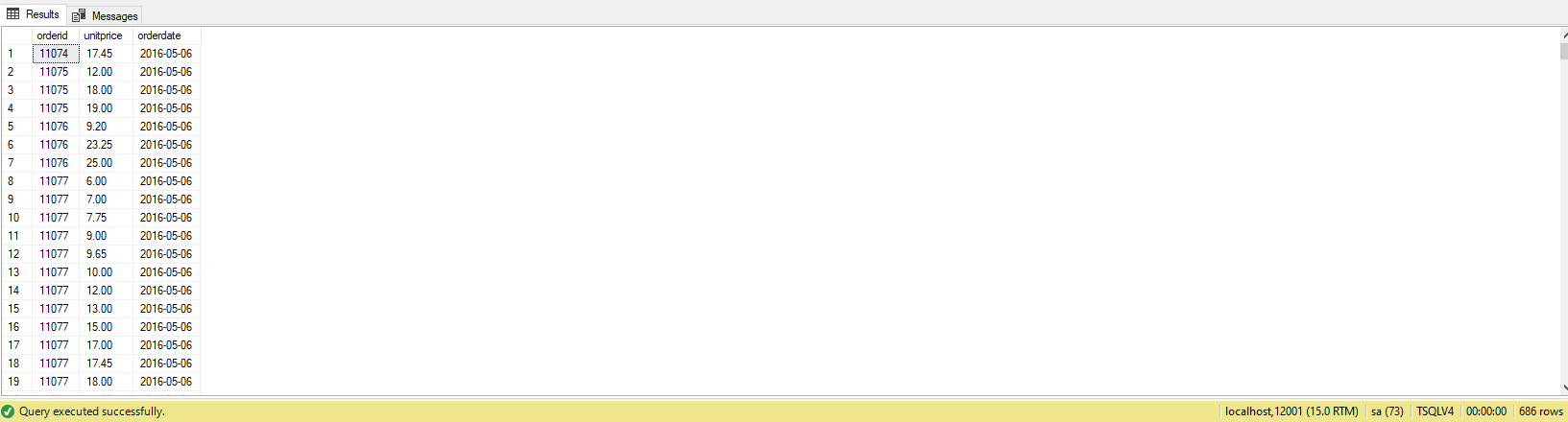
O.unitprice,

P.orderid,

P.orderdate

ORDER BY P.orderdate DESC;

--FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution with output (686 objects returned)**

USE TSQLV4;

SELECT O.orderid,

O.unitprice,

P.orderdate

FROM dbo.Orderdetails AS O

INNER JOIN dbo.Orders AS P

ON P.orderid = O.orderid

AND YEAR(P.orderdate) = 2016

GROUP BY O.orderid,

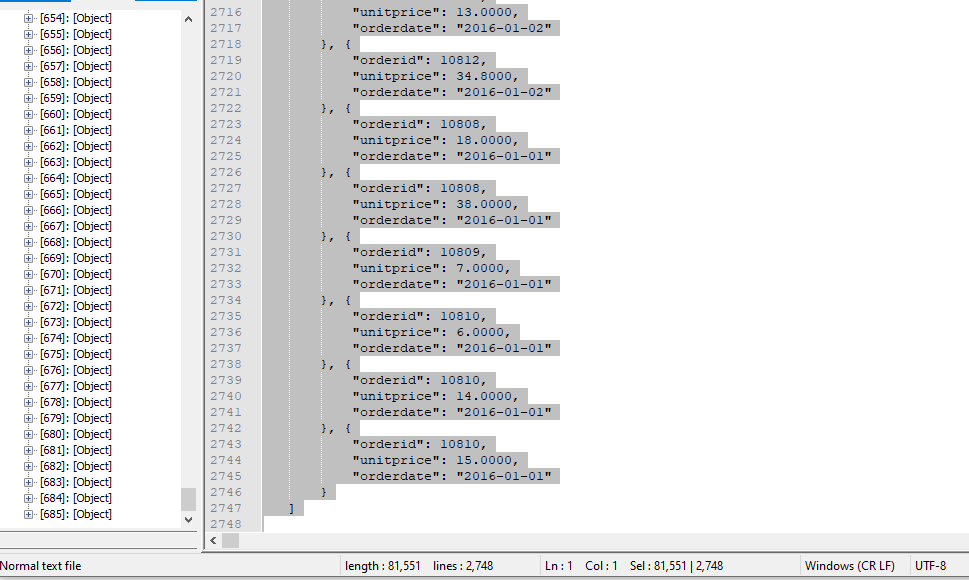
O.unitprice,

P.orderid,

P.orderdate

ORDER BY P.orderdate DESC

FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Medium Query 2**

Performs an Inner join and returns all distinct customer/employee ids and fright where the city is London

Use NorthwindsTSQLV6 database and Sales.Customer and Sales.Order tables

**Standard View**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Sales.Customer | CustomerId  CustomerCity | ASC  ASC |
| Sales.Order | EmployeeId  Freight | ASC  ASC |

**Sample Relational solution with output (1 row)**

USE Northwinds2020TSQLV6;

SELECT DISTINCT

A.CustomerId AS [Customer ID],

A.CustomerCity AS [Customer City],

B.EmployeeId AS [Employee ID],

MAX(B.Freight) AS [MAX Freight]

FROM Sales.[Customer] AS A

INNER JOIN Sales.[Order] AS B

ON A.CustomerId = B.EmployeeId

WHERE B.Freight > 20

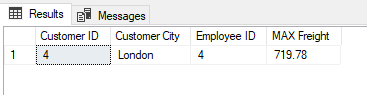
AND A.CustomerCity = 'London'

GROUP BY A.CustomerId,

A.CustomerCity,

B.EmployeeId;

--FOR JSON PATH, ROOT('CustomerId'), INCLUDE\_NULL\_VALUES;



**Sample Relational solution with output (1 Object)**

USE Northwinds2020TSQLV6;

SELECT DISTINCT

A.CustomerId AS [Customer ID],

A.CustomerCity AS [Customer City],

B.EmployeeId AS [Employee ID],

MAX(B.Freight) AS [MAX Freight]

FROM Sales.[Customer] AS A

INNER JOIN Sales.[Order] AS B

ON A.CustomerId = B.EmployeeId

WHERE B.Freight > 20

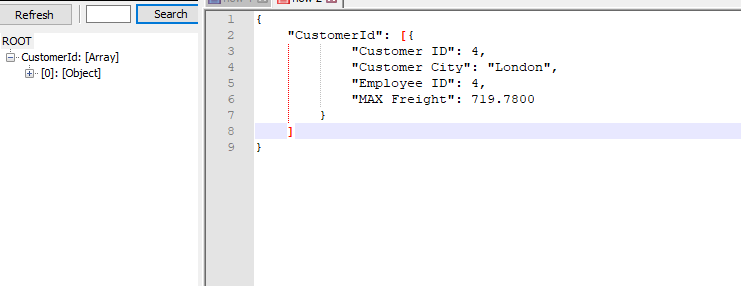
AND A.CustomerCity = 'London'

GROUP BY A.CustomerId,

A.CustomerCity,

B.EmployeeId

FOR JSON PATH, ROOT('CustomerId'), INCLUDE\_NULL\_VALUES;



**Medium Query 3**

Performs an Inner join and returns credit card id, number and the business entity associated with it ordered by the credit card id which is greater than 18000

Use AdventureWorks2017 database and Sales.CreditCard and Sales.PersonCreditCard tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Sales.CreditCard | CreditCardID  CardNumber | ASC |
| Sales.PersonCreditCard | BusinessEntityID | ASC |

**Sample Relational solution and output (1186 rows)**

USE AdventureWorks2017;

SELECT MIN(C.CreditCardID) AS [Min card],

C.CardNumber,

B.BusinessEntityID

FROM Sales.CreditCard AS C

INNER JOIN Sales.PersonCreditCard AS B

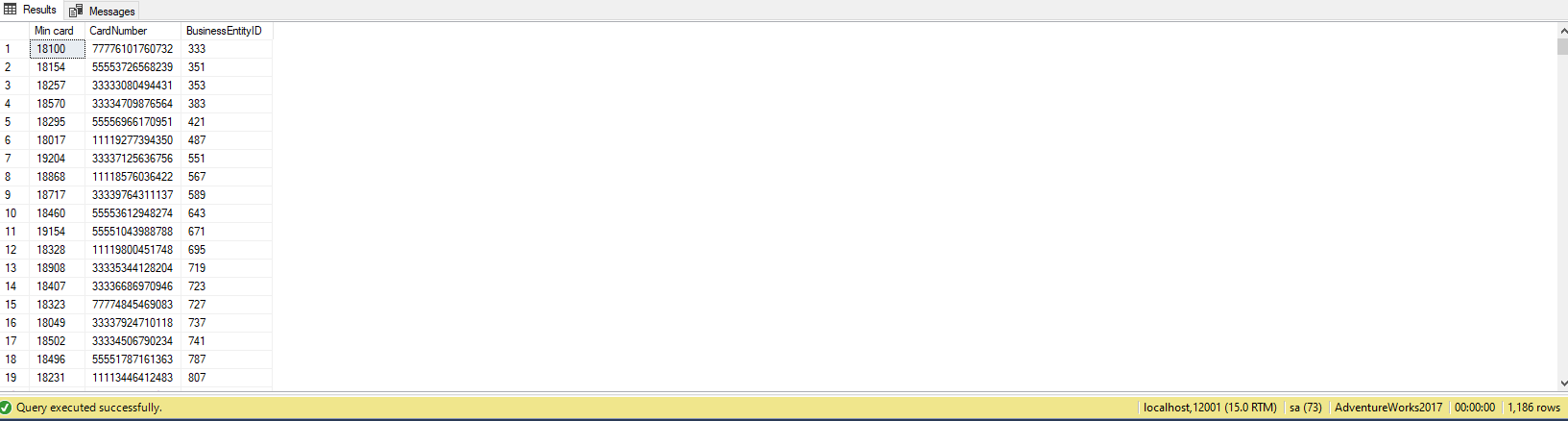
ON B.CreditCardID = C.CreditCardID

AND C.CreditCardID > 18000

GROUP BY C.CardNumber,

B.BusinessEntityID;

--FOR JSON PATH, ROOT('CreditCardID'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution and output (1186 Objects)**

USE AdventureWorks2017;

SELECT MIN(C.CreditCardID) AS [Min card],

C.CardNumber,

B.BusinessEntityID

FROM Sales.CreditCard AS C

INNER JOIN Sales.PersonCreditCard AS B

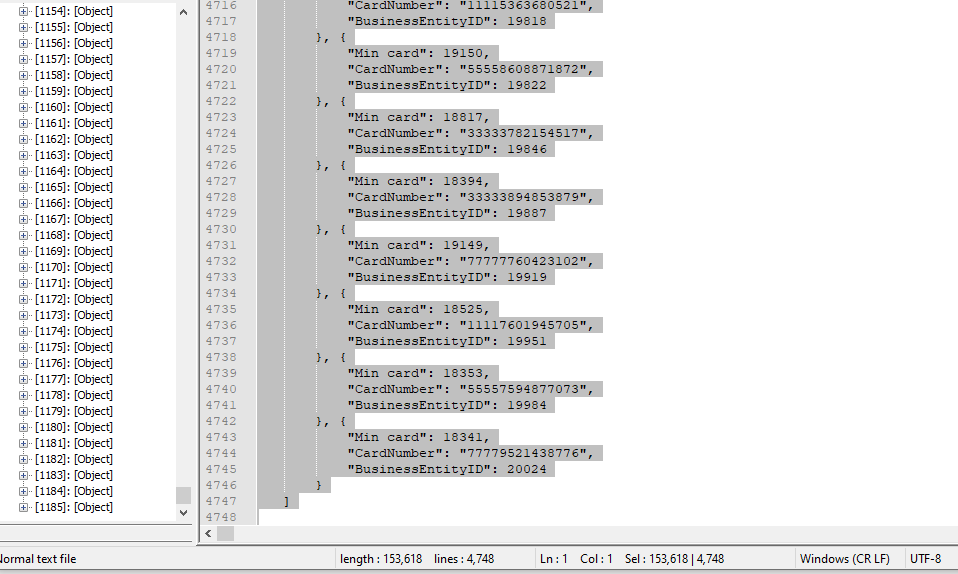
ON B.CreditCardID = C.CreditCardID

AND C.CreditCardID > 18000

GROUP BY C.CardNumber,

B.BusinessEntityID

FOR JSON PATH, ROOT('CreditCardID'), INCLUDE\_NULL\_VALUES;



**Medium Query 4**

Performs an Inner join and returns the maximum income and max geo key from each customer id between 15000 – 15500

Use AdventureWorksDW2017 database and dbo.DimGeography and dbo.DimCustomer tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Dbo.DimCustomer | GeographyKey  CustomerKey  YearlyIncome | ASC  ASC  ASC |
| Dbo.DimGeography | City  CountryRegionCode | ASC  ASC |

**Sample relational solution with output (499 rows returned)**

USE AdventureWorksDW2017;

SELECT MAX(C.GeographyKey) AS [Max Geo],

C.CustomerKey,

C.YearlyIncome AS [Max Income],

D.City,

D.CountryRegionCode

FROM dbo.DimCustomer AS C

INNER JOIN dbo.DimGeography AS D

ON C.GeographyKey = D.GeographyKey

WHERE C.CustomerKey > 15000

AND C.CustomerKey < 15500

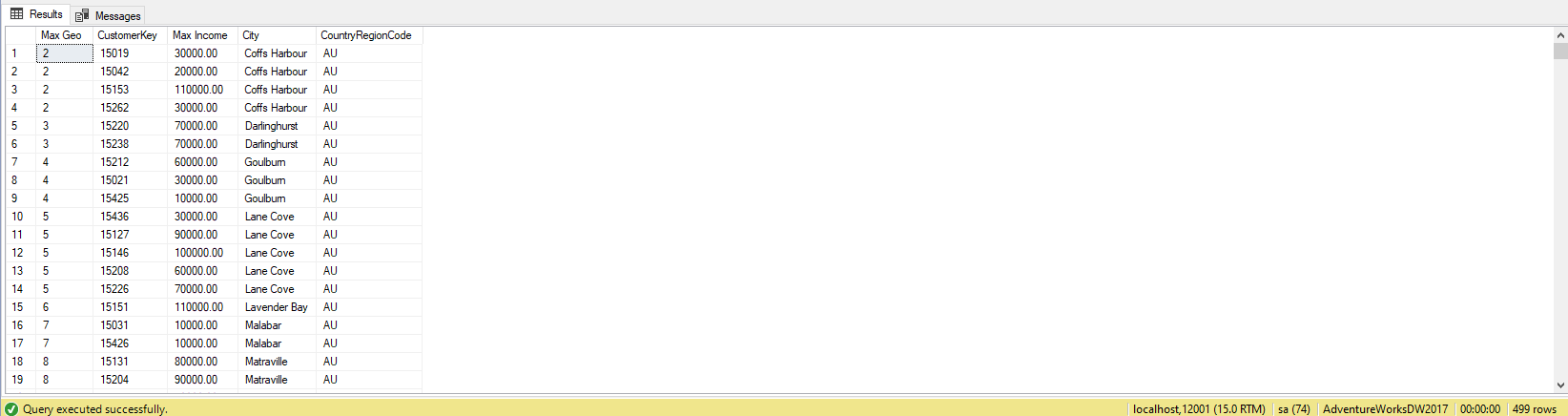
GROUP BY C.CustomerKey,

C.YearlyIncome,

D.City,

D.CountryRegionCode;

--FOR JSON PATH, ROOT('CreditCardID'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution with output (499 Objects returned)**

USE AdventureWorksDW2017;

SELECT MAX(C.GeographyKey) AS [Max Geo],

C.CustomerKey,

C.YearlyIncome AS [Max Income],

D.City,

D.CountryRegionCode

FROM dbo.DimCustomer AS C

INNER JOIN dbo.DimGeography AS D

ON C.GeographyKey = D.GeographyKey

WHERE C.CustomerKey > 15000

AND C.CustomerKey < 15500

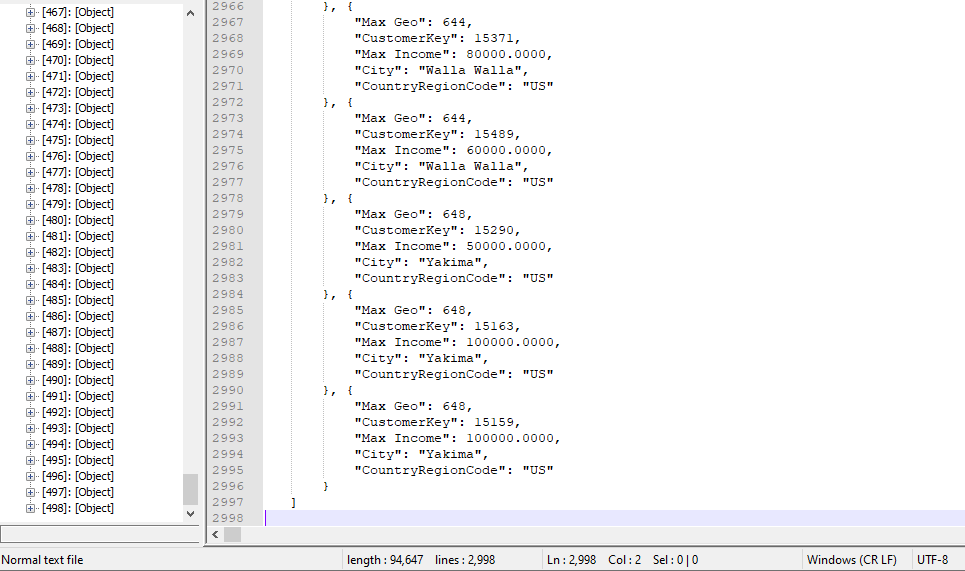
GROUP BY C.CustomerKey,

C.YearlyIncome,

D.City,

D.CountryRegionCode

FOR JSON PATH, ROOT('CreditCardID'), INCLUDE\_NULL\_VALUES;



**Medium Query 5**

Performs an left join and returns every order made by customer 90

Use TSQLV6 database and Sales.Orders and dbo.Orders

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Dbo.Order | Orderid  custid  freight | ASC  ASC  ASC |
| Sales.Orders | shippeddate | ASC |

**Sample Relational solution and output (Returns 7 rows)**

USE TSQLV4;

SELECT O.orderid,

O.custid,

SUM(O.freight) AS [Summed Freight],

S.shippeddate

FROM dbo.Orders AS O

LEFT JOIN Sales.Orders AS S

ON S.custid = O.custid

AND S.orderid = O.orderid

WHERE O.custid = 90

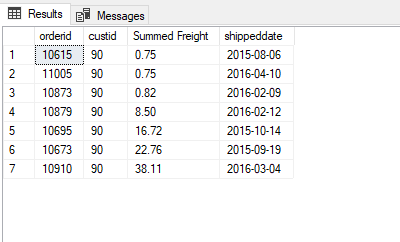
GROUP BY O.orderid,

O.custid,

S.shippeddate

ORDER BY [Summed Freight];

--FOR JSON PATH, ROOT('custid'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution and output (Returns 7 objects)**

USE TSQLV4;

SELECT O.orderid,

O.custid,

SUM(O.freight) AS [Summed Freight],

S.shippeddate

FROM dbo.Orders AS O

LEFT JOIN Sales.Orders AS S

ON S.custid = O.custid

AND S.orderid = O.orderid

WHERE O.custid = 90

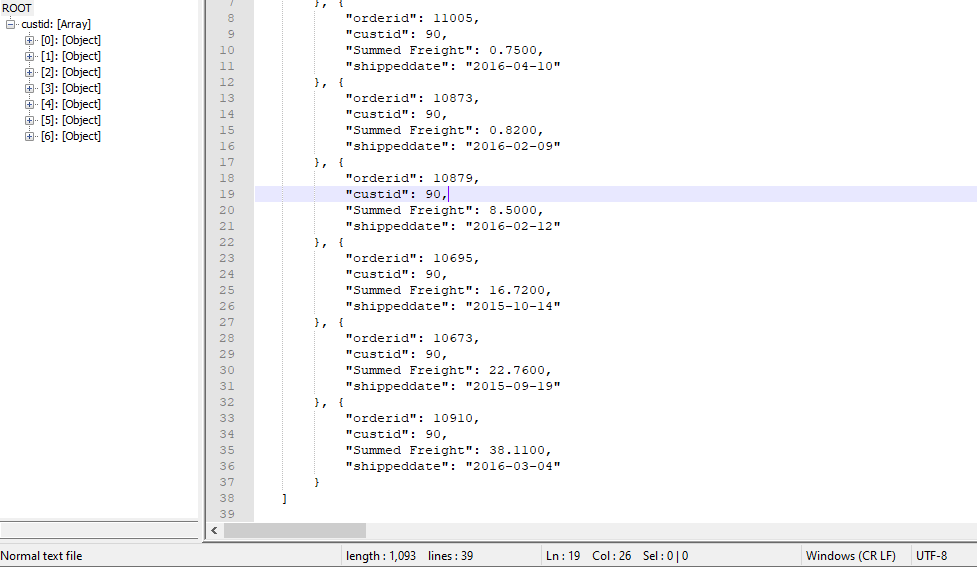
GROUP BY O.orderid,

O.custid,

S.shippeddate

ORDER BY [Summed Freight]

FOR JSON PATH, ROOT('custid'), INCLUDE\_NULL\_VALUES;



**Medium Query 6**

Performs an inner join and returns order id, customer id, the maximum freight for that order with its destination

Use TSQLV6 database with Sales.Orders and Scratch.Orders tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Sales.Orders | Orderid  custid  freight | ASC  ASC  ASC |
| Scratch.Orders | shipto | ASC |

**Sample relational solution and output (Returns 48 Rows)**

USE TSQLV4;

SELECT O.orderid,

O.custid,

MAX(O.freight) AS [MAX Freight],

S.shipname AS [Ship To]

FROM Sales.Orders AS O

INNER JOIN Scratch.Orders AS S

ON O.orderid = S.orderid

WHERE O.orderid IS NOT NULL

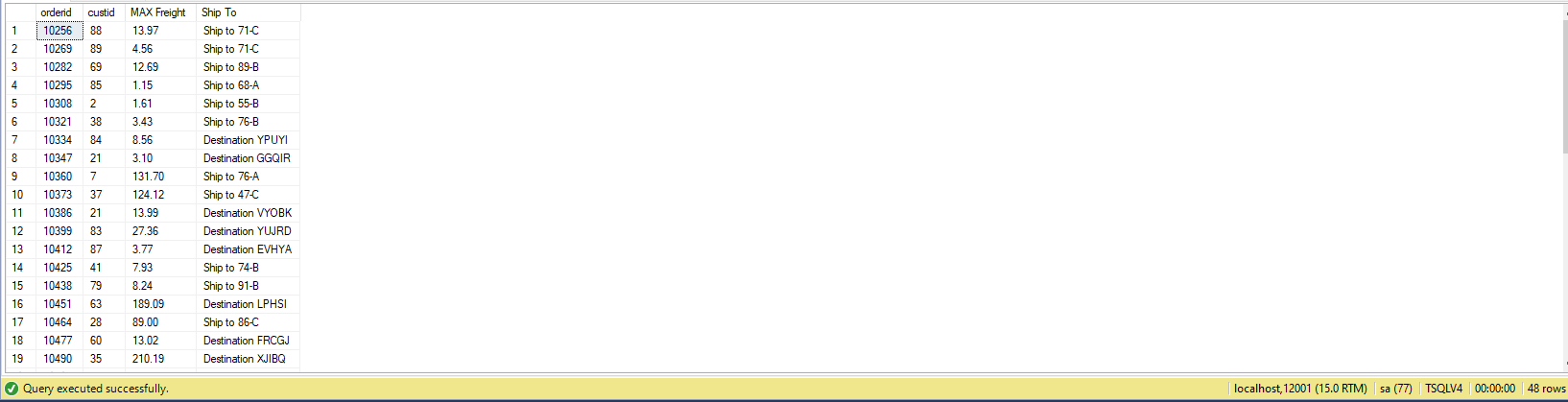
GROUP BY O.orderid,

O.custid,

S.shipname

ORDER BY O.orderid;

--FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution and output (Returns 48 Objects)**

USE TSQLV4;

SELECT O.orderid,

O.custid,

MAX(O.freight) AS [MAX Freight],

S.shipname AS [Ship To]

FROM Sales.Orders AS O

INNER JOIN Scratch.Orders AS S

ON O.orderid = S.orderid

WHERE O.orderid IS NOT NULL

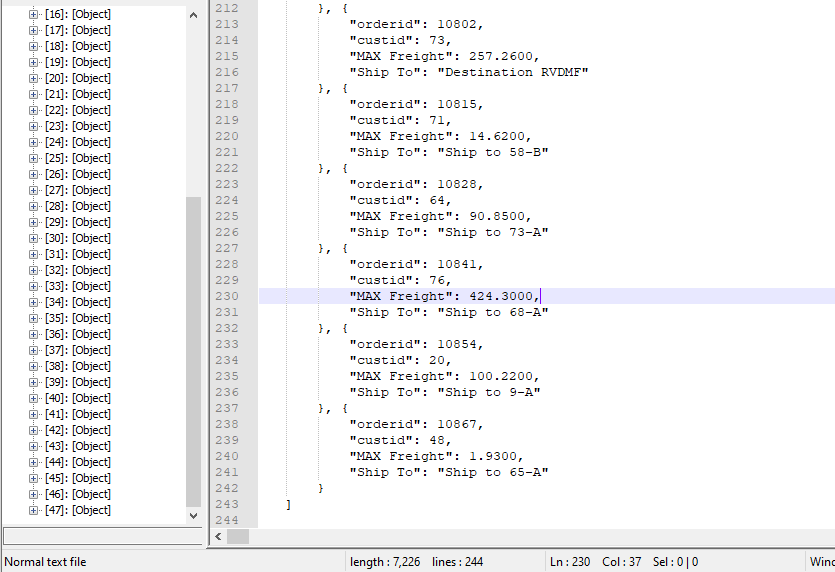
GROUP BY O.orderid,

O.custid,

S.shipname

ORDER BY O.orderid

FOR JSON PATH, ROOT('custid'), INCLUDE\_NULL\_VALUES;



**Medium Query 7**

Performs an inner join and returns customer's data with minimum freight where the year 2014 and customer id > 60 which is ordered by the customer id

Use NorthwindsTSQLV6 Database with Sales.Customer and Sales.Order tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Sales.Orders | CustomerID  CustomerCompanyName  CustomerPostalCode | ASC  ASC  ASC |
| Sales.[Order] | EmployeeID | ASC |

**Sample relational solution and output (Returns 57 rows)**

USE Northwinds2020TSQLV6;

SELECT C.CustomerId,

C.CustomerCompanyName,

C.CustomerPostalCode,

E.EmployeeId,

MIN(E.Freight) AS [MIN Freight]

FROM Sales.Customer AS C

INNER JOIN Sales.[Order] AS E

ON E.CustomerId = C.CustomerId

WHERE YEAR(E.OrderDate) = '2014'

AND E.CustomerId > 60

GROUP BY C.CustomerId,

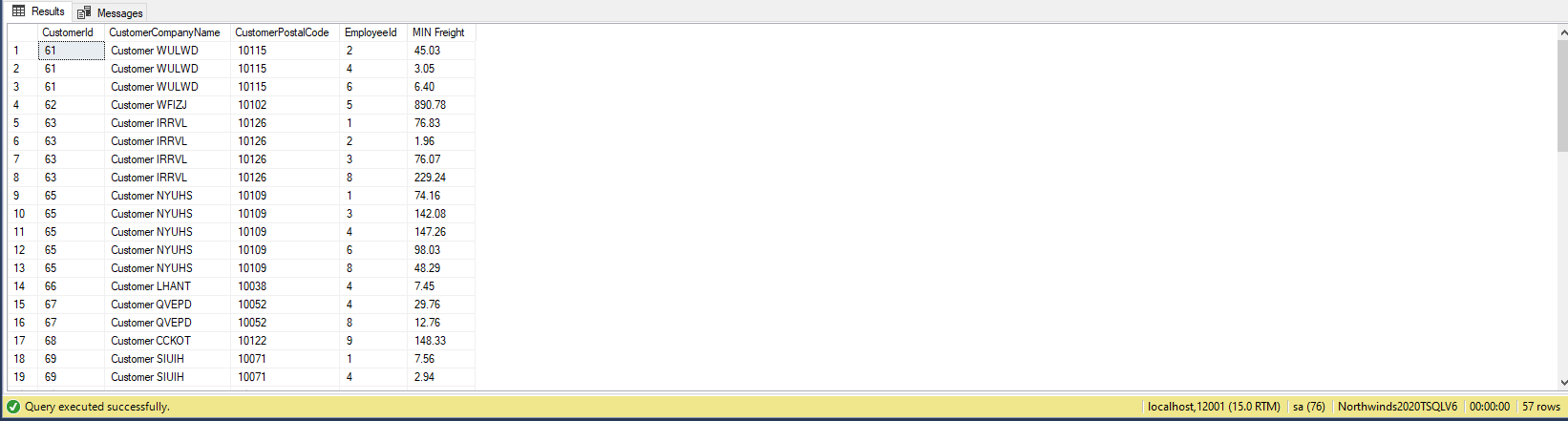
C.CustomerCompanyName,

C.CustomerPostalCode,

E.EmployeeId

ORDER BY C.CustomerId;

--FOR JSON PATH, ROOT('customerid'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution and output (Returns 57 Objects)**

USE Northwinds2020TSQLV6;

SELECT C.CustomerId,

C.CustomerCompanyName,

C.CustomerPostalCode,

E.EmployeeId,

MIN(E.Freight) AS [MIN Freight]

FROM Sales.Customer AS C

INNER JOIN Sales.[Order] AS E

ON E.CustomerId = C.CustomerId

WHERE YEAR(E.OrderDate) = '2014'

AND E.CustomerId > 60

GROUP BY C.CustomerId,

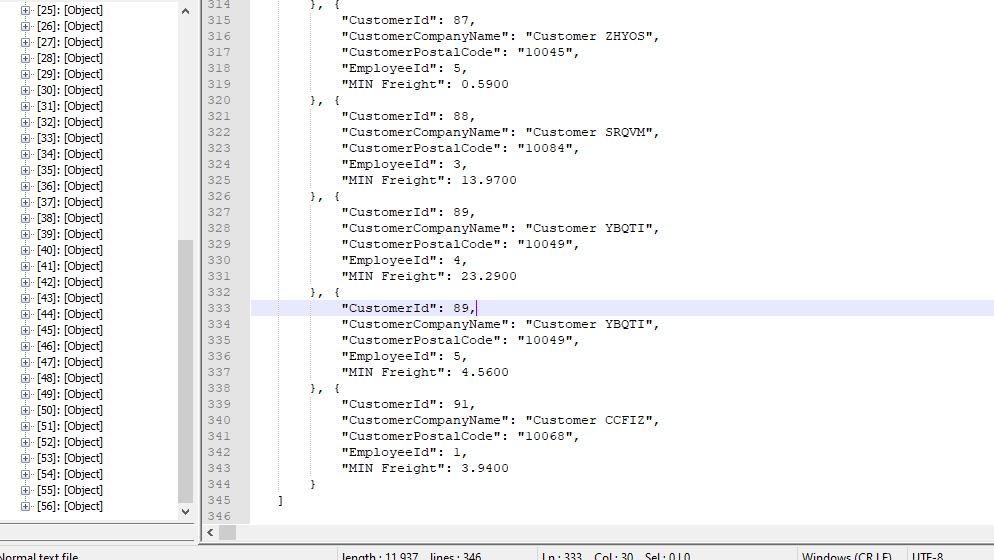
C.CustomerCompanyName,

C.CustomerPostalCode,

E.EmployeeId

ORDER BY C.CustomerId

FOR JSON PATH, ROOT('custid'), INCLUDE\_NULL\_VALUES;



**Medium Query 8**

Performs an inner join and returns the employee key, quota, the maximum base pay where the employee was hired after 2011

Use AdventureWorksDW2017 database with dbo.DimEmployee and dbo.FactSalesQuota tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Dbo.FactSalesQuota | EmployeeKey  SalesAmountQuota | ASC  ASC |
| Dbo.DimEmployee | HireDate  BaseRate | ASC  ASC |

**Sample relational solution and output (Returns 19 rows)**

USE AdventureWorksDW2017;

SELECT A.EmployeeKey,

A.SalesAmountQuota,

B.HireDate,

MAX(B.BaseRate) AS [Maximum Rate]

FROM dbo.FactSalesQuota AS A

INNER JOIN dbo.DimEmployee AS B

ON B.EmployeeKey = A.EmployeeKey

WHERE YEAR(B.HireDate) > 2011

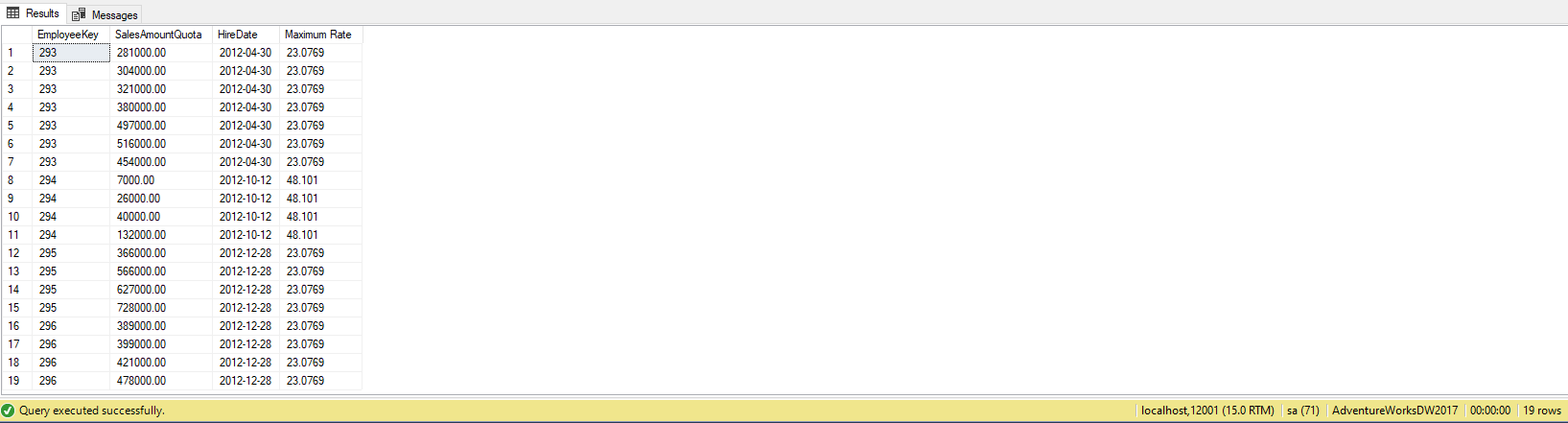
GROUP BY A.EmployeeKey,

A.SalesAmountQuota,

B.HireDate

ORDER BY B.HireDate;

--FOR JSON PATH, ROOT('EmployeeKey'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution and output (Returns 19 Objects)**

USE AdventureWorksDW2017;

SELECT A.EmployeeKey,

A.SalesAmountQuota,

B.HireDate,

MAX(B.BaseRate) AS [Maximum Rate]

FROM dbo.FactSalesQuota AS A

INNER JOIN dbo.DimEmployee AS B

ON B.EmployeeKey = A.EmployeeKey

WHERE YEAR(B.HireDate) > 2011

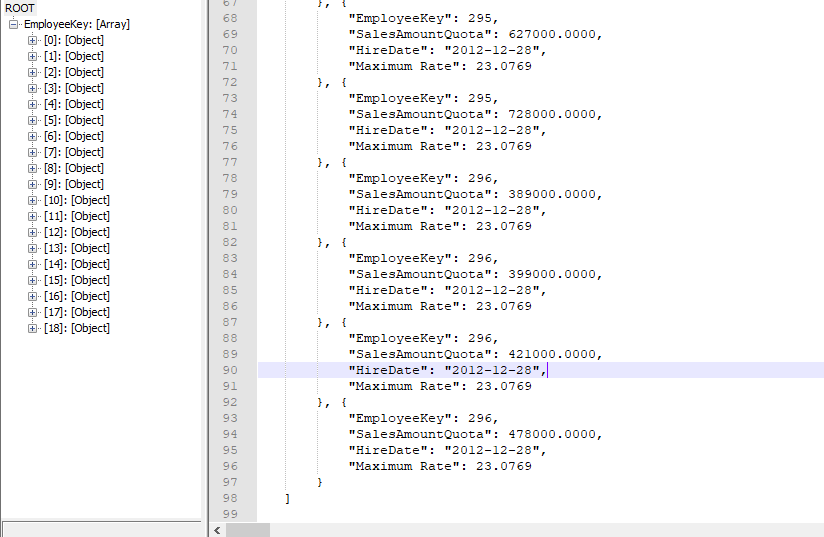
GROUP BY A.EmployeeKey,

A.SalesAmountQuota,

B.HireDate

ORDER BY B.HireDate

FOR JSON PATH, ROOT('EmployeeKey'), INCLUDE\_NULL\_VALUES;



**Complex Query 1**

Scalar function that returns the maximum customer id. Query performs a inner join twice and returns the max customer id, customer's company name and the orderdate

Use TSQLV4 database with Sales.Customers, Scratch.Orders, and Sales.Orders tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Sales.Customers | CompanyName  Custid | ASC  ASC |
| Sales.Orders | orderdate | ASC |
| Scratch.Orders | Custid | ASC |

**Sample relational solution with output (Returns 48 rows)**

USE TSQLV4;

DROP FUNCTION IF EXISTS dbo.CustomerInfo;

GO

CREATE FUNCTION dbo.CustomerInfo

(

@Custid AS INT

)

RETURNS INT

AS

BEGIN

RETURN

(

SELECT MAX(@Custid) FROM Sales.Customers

);

END;

GO

USE TSQLV4;

SELECT dbo.CustomerInfo(A.custid) AS [Max Customer ID],

A.companyname AS [Company Name],

B.orderdate AS [Order Date]

FROM Sales.Customers AS A

INNER JOIN Sales.Orders AS B

ON A.custid = B.custid

INNER JOIN Scratch.Orders AS C

ON B.orderid = C.orderid

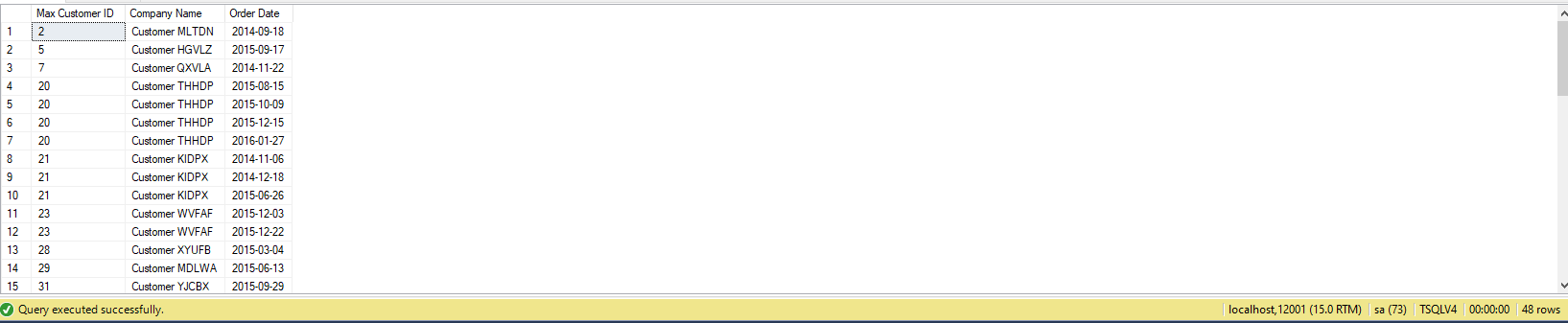
GROUP BY dbo.CustomerInfo(A.custid),

A.companyname,

B.orderdate,

A.custid

ORDER BY A.custid;



**Sample JSON solution with output (Returns 48 Objects)**

USE TSQLV4;

DROP FUNCTION IF EXISTS dbo.CustomerInfo;

GO

CREATE FUNCTION dbo.CustomerInfo

(

@Custid AS INT

)

RETURNS INT

AS

BEGIN

RETURN

(

SELECT MAX(@Custid) FROM Sales.Customers

);

END;

GO

USE TSQLV4;

SELECT dbo.CustomerInfo(A.custid) AS [Max Customer ID],

A.companyname AS [Company Name],

B.orderdate AS [Order Date]

FROM Sales.Customers AS A

INNER JOIN Sales.Orders AS B

ON A.custid = B.custid

INNER JOIN Scratch.Orders AS C

ON B.orderid = C.orderid

GROUP BY dbo.CustomerInfo(A.custid),

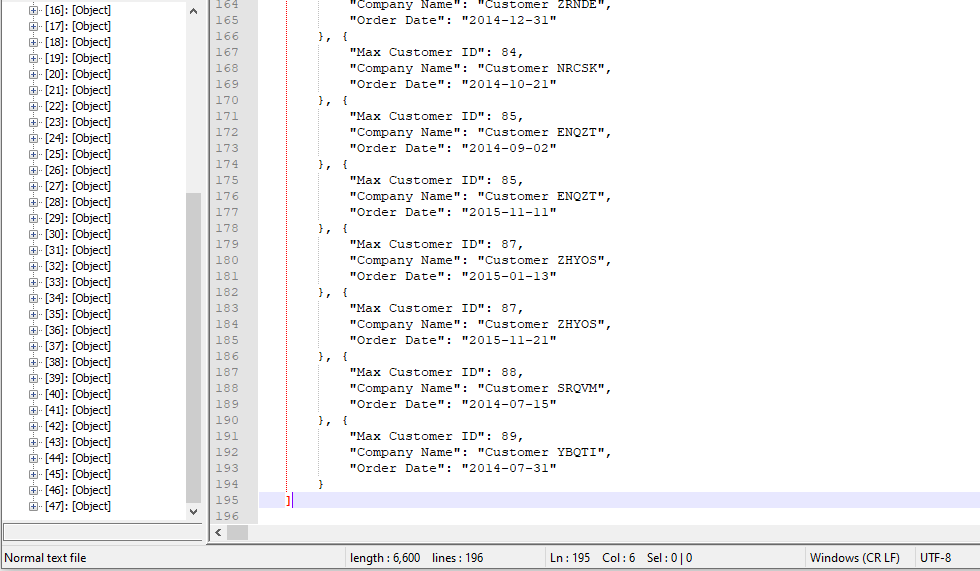
A.companyname,

B.orderdate,

A.custid

ORDER BY A.custid

FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Complex Query 2**

Scalar function that returns the minimum customer id. Query performs a inner join twice and returns the min customer id, customer's company name and the orderdate

Use TSQLV4 database with Sales.Customers, Scratch.Orders, and Sales.Orders tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Sales.Customers | CompanyName  Custid | ASC  ASC |
| Sales.Orders | orderdate | ASC |
| Scratch.Orders | Custid | ASC |

**Sample relational solution with output (Returns 48 rows)**

USE TSQLV4;

DROP FUNCTION IF EXISTS dbo.MinCustomerInfo;

GO

CREATE FUNCTION dbo.MinCustomerInfo

(

@Custid AS INT

)

RETURNS INT

AS

BEGIN

RETURN

(

SELECT MIN(@Custid) FROM Sales.Customers

);

END;

GO

USE TSQLV4;

SELECT dbo.MinCustomerInfo(A.custid) AS [Min Customer ID],

A.companyname AS [Company Name],

B.orderdate AS [Order Date]

FROM Sales.Customers AS A

INNER JOIN Sales.Orders AS B

ON A.custid = B.custid

INNER JOIN Scratch.Orders AS C

ON B.orderid = C.orderid

GROUP BY dbo.MinCustomerInfo(A.custid),

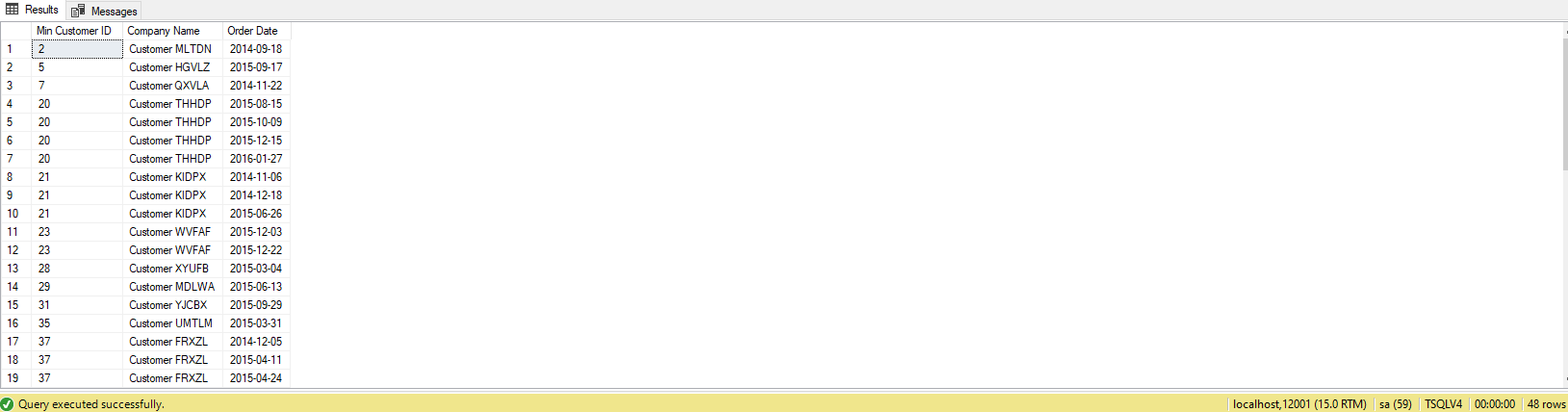
A.companyname,

B.orderdate,

A.custid

ORDER BY A.custid;

--FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution with output (Returns 48 Objects)**

USE TSQLV4;

DROP FUNCTION IF EXISTS dbo.MinCustomerInfo;

GO

CREATE FUNCTION dbo.MinCustomerInfo

(

@Custid AS INT

)

RETURNS INT

AS

BEGIN

RETURN

(

SELECT MIN(@Custid) FROM Sales.Customers

);

END;

GO

USE TSQLV4;

SELECT dbo.MinCustomerInfo(A.custid) AS [Min Customer ID],

A.companyname AS [Company Name],

B.orderdate AS [Order Date]

FROM Sales.Customers AS A

INNER JOIN Sales.Orders AS B

ON A.custid = B.custid

INNER JOIN Scratch.Orders AS C

ON B.orderid = C.orderid

GROUP BY dbo.MinCustomerInfo(A.custid),

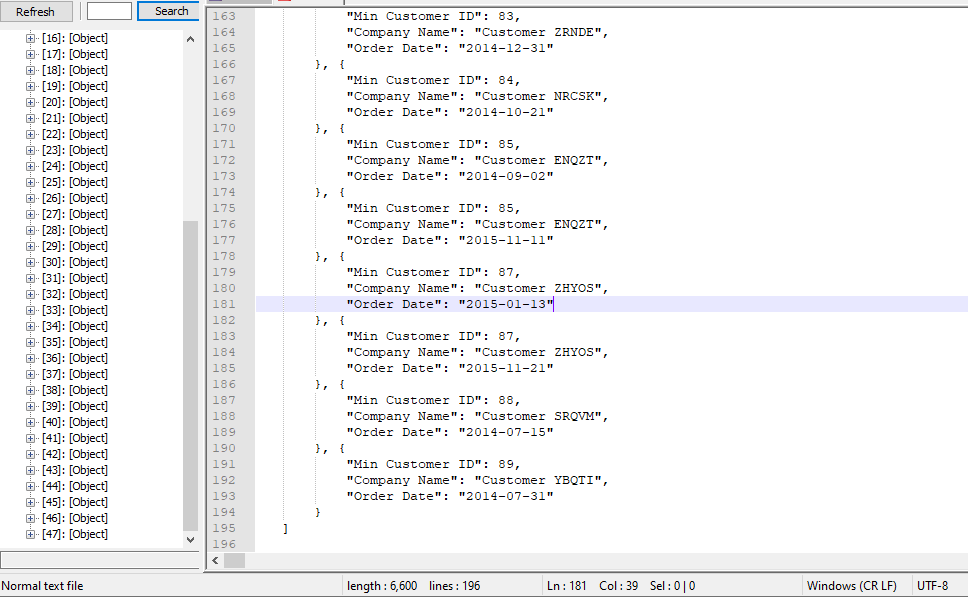
A.companyname,

B.orderdate,

A.custid

ORDER BY A.custid

FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Complex Query 3**

Scalar function returns the minimum number of issues from the company. The query returns the English product name and its category with its size, the average number of calls they get where issues are less than 3

Use AdventureWorksDW2017 database and dbo.DimProduct, dbo.DimProductSubcategory, dbo.FactCallCenter tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Dbo.DimProduct | EnglishProductName  Size | ASC  ASC |
| Dbo.DimProductSubcategory | EnglishProductSubcategoryName | ASC  ASC |
| Dbo.FactCallCenter | IssuesRaised | ASC |

**Sample relational solution with output (Returns 633 rows)**

USE AdventureWorksDW2017;

DROP FUNCTION IF EXISTS dbo.MinProductIssueFunction;

GO

CREATE FUNCTION dbo.MinProductIssueFunction

(

@issues AS SMALLINT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MIN(@issues) FROM dbo.FactCallCenter

);

END;

GO

USE AdventureWorksDW2017;

SELECT A.EnglishProductName,

A.Size,

B.EnglishProductSubcategoryName,

dbo.MinProductIssueFunction(C.IssuesRaised) AS [Number of Issues],

AVG(C.Calls) AS [Average Calls]

FROM dbo.DimProduct AS A

INNER JOIN dbo.DimProductSubcategory AS B

ON B.ProductSubcategoryKey = A.ProductSubcategoryKey

INNER JOIN dbo.FactCallCenter AS C

ON A.Size IS NOT NULL

AND dbo.MinProductIssueFunction(C.IssuesRaised) < 3

GROUP BY dbo.MinProductIssueFunction(C.IssuesRaised),

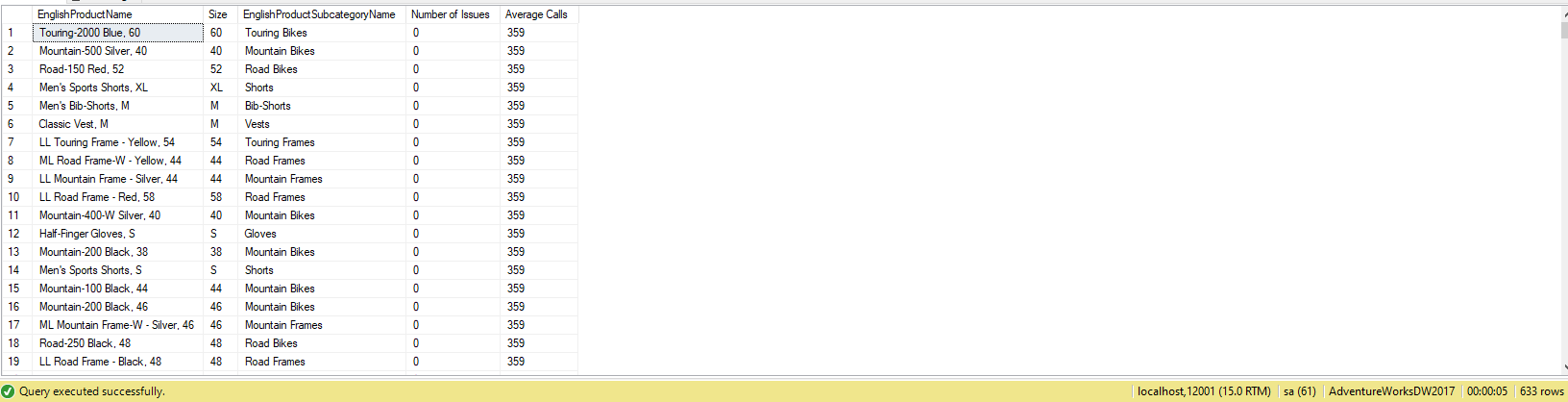
A.EnglishProductName,

A.Size,

B.EnglishProductSubcategoryName

ORDER BY [Number of Issues];

--FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution with output (Returns 633 Objects)**

USE AdventureWorksDW2017;

DROP FUNCTION IF EXISTS dbo.MinProductIssueFunction;

GO

CREATE FUNCTION dbo.MinProductIssueFunction

(

@issues AS SMALLINT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MIN(@issues) FROM dbo.FactCallCenter

);

END;

GO

USE AdventureWorksDW2017;

SELECT A.EnglishProductName,

A.Size,

B.EnglishProductSubcategoryName,

dbo.MinProductIssueFunction(C.IssuesRaised) AS [Number of Issues],

AVG(C.Calls) AS [Average Calls]

FROM dbo.DimProduct AS A

INNER JOIN dbo.DimProductSubcategory AS B

ON B.ProductSubcategoryKey = A.ProductSubcategoryKey

INNER JOIN dbo.FactCallCenter AS C

ON A.Size IS NOT NULL

AND dbo.MinProductIssueFunction(C.IssuesRaised) < 3

GROUP BY dbo.MinProductIssueFunction(C.IssuesRaised),

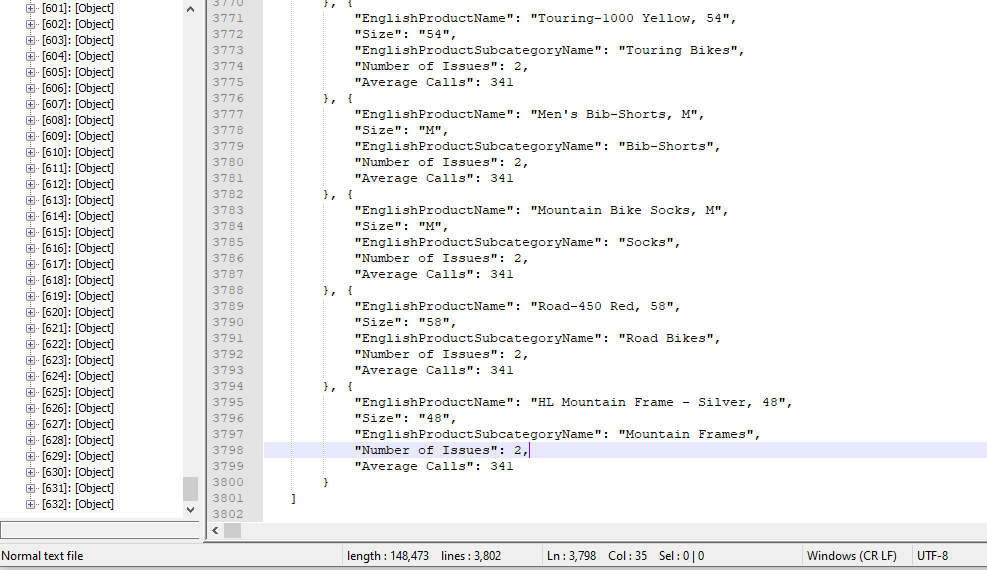
A.EnglishProductName,

A.Size,

B.EnglishProductSubcategoryName

ORDER BY [Number of Issues]

FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Complex Query 4**

Scalar function returns the maximum number of issues from the company. The query returns the English product name and its category with its size, the average number of calls they get where issues are greater than 2

Use AdventureWorksDW2017 database and dbo.DimProduct, dbo.DimProductSubcategory, dbo.FactCallCenter tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Dbo.DimProduct | EnglishProductName  Size | ASC  ASC |
| Dbo.DimProductSubcategory | EnglishProductSubcategoryName | ASC |
| Dbo.FactCallCenter | IssuesRaised | ASC |

**Sample relational solution with output (Returns 210 rows)**

USE AdventureWorksDW2017;

DROP FUNCTION IF EXISTS dbo.MaxProductIssueFunction;

GO

CREATE FUNCTION dbo.MaxProductIssueFunction

(

@issues AS SMALLINT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MAX(@issues) FROM dbo.FactCallCenter

);

END;

GO

USE AdventureWorksDW2017;

SELECT A.EnglishProductName,

A.Size,

B.EnglishProductSubcategoryName,

dbo.MaxProductIssueFunction(C.IssuesRaised) AS [Number of Issues],

AVG(C.Calls) AS [Average Calls]

FROM dbo.DimProduct AS A

INNER JOIN dbo.DimProductSubcategory AS B

ON B.ProductSubcategoryKey = A.ProductSubcategoryKey

INNER JOIN dbo.FactCallCenter AS C

ON A.Size IS NOT NULL

AND dbo.MaxProductIssueFunction(C.IssuesRaised) > 2

GROUP BY dbo.MaxProductIssueFunction(C.IssuesRaised),

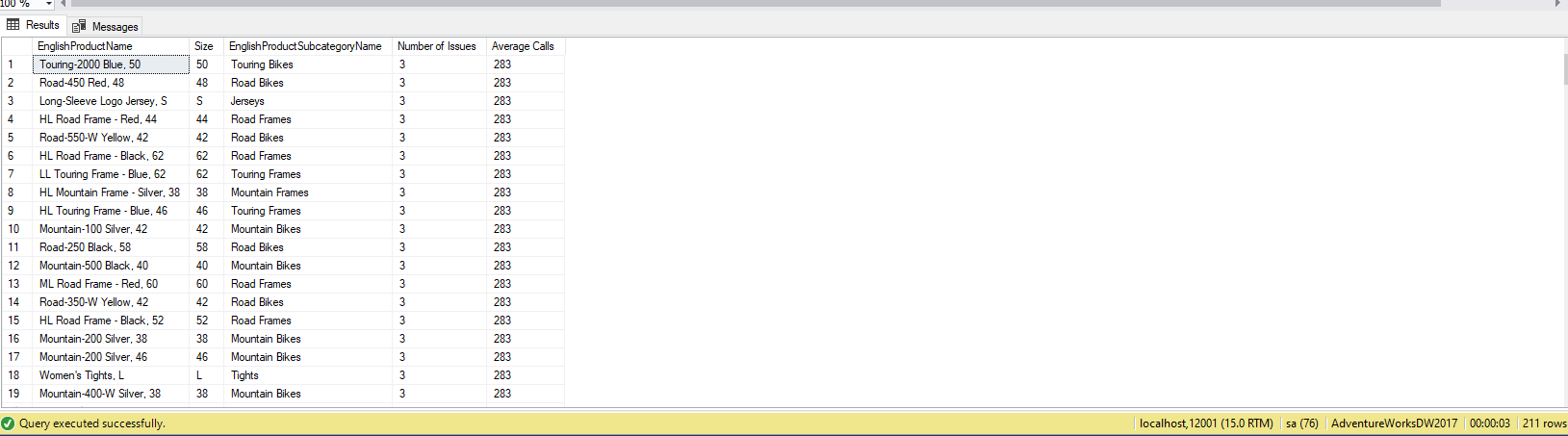
A.EnglishProductName,

A.Size,

B.EnglishProductSubcategoryName

ORDER BY [Number of Issues];

--FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution with output (Returns 210 objects)**

USE AdventureWorksDW2017;

DROP FUNCTION IF EXISTS dbo.MaxProductIssueFunction;

GO

CREATE FUNCTION dbo.MaxProductIssueFunction

(

@issues AS SMALLINT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MAX(@issues) FROM dbo.FactCallCenter

);

END;

GO

USE AdventureWorksDW2017;

SELECT A.EnglishProductName,

A.Size,

B.EnglishProductSubcategoryName,

dbo.MaxProductIssueFunction(C.IssuesRaised) AS [Number of Issues],

AVG(C.Calls) AS [Average Calls]

FROM dbo.DimProduct AS A

INNER JOIN dbo.DimProductSubcategory AS B

ON B.ProductSubcategoryKey = A.ProductSubcategoryKey

INNER JOIN dbo.FactCallCenter AS C

ON A.Size IS NOT NULL

AND dbo.MaxProductIssueFunction(C.IssuesRaised) > 2

GROUP BY dbo.MaxProductIssueFunction(C.IssuesRaised),

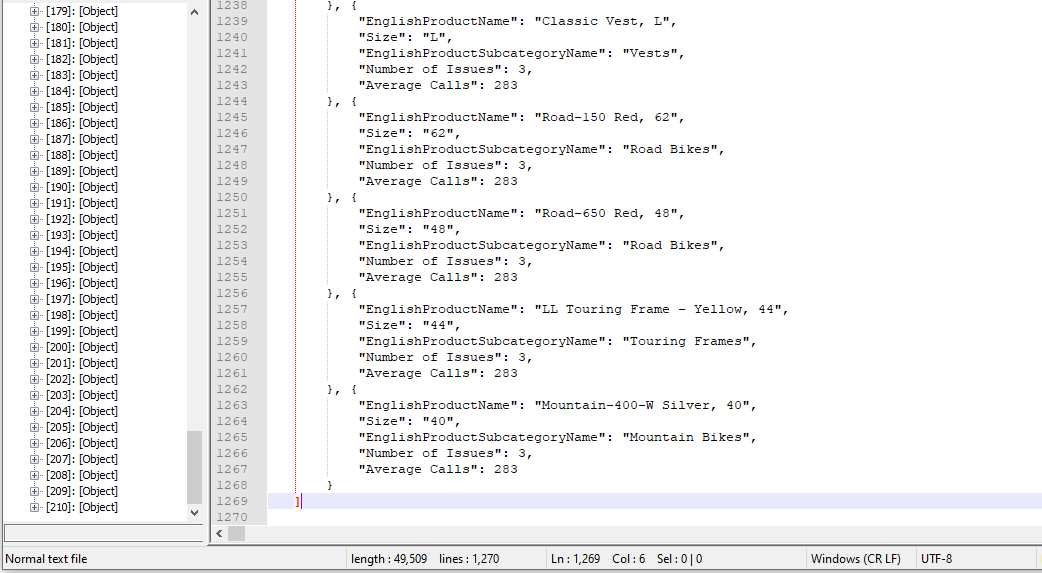
A.EnglishProductName,

A.Size,

B.EnglishProductSubcategoryName

ORDER BY [Number of Issues]

FOR JSON PATH, ROOT('orderid'), INCLUDE\_NULL\_VALUES;



**Complex Query 5**

Scalar function returns the minimum Email promotions. Query returns top 50 email address, password hash, first and last name where minimum promotions are less than 2

Use AdventureWorks2017 database with Person.EmailAddress, Person.BusinessEntity, and Person.Password tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Person.EmailAddress | EmailAddress | ASC |
| Person.Password | PasswordHash | ASC |
| Person.Person | FirstName  LastName | ASC  ASC |

**Sample Relational Solution with output (returns 50 rows)**

USE AdventureWorks2017;

DROP FUNCTION IF EXISTS dbo.MinPromotion;

GO

CREATE FUNCTION dbo.MinPromotion

(

@Promotion AS INT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MIN(@Promotion) FROM Person.Person

);

END;

GO

USE AdventureWorks2017;

SELECT TOP 50

A.EmailAddress,

B.PasswordHash,

C.FirstName,

C.LastName,

dbo.MinPromotion(C.EmailPromotion) AS [Minimum Promotions]

FROM Person.EmailAddress AS A

INNER JOIN Person.Password AS B

ON B.BusinessEntityID = A.BusinessEntityID

INNER JOIN Person.Person AS C

ON C.BusinessEntityID = A.BusinessEntityID

WHERE dbo.MinPromotion(C.EmailPromotion) < 2

GROUP BY dbo.MinPromotion(C.EmailPromotion),

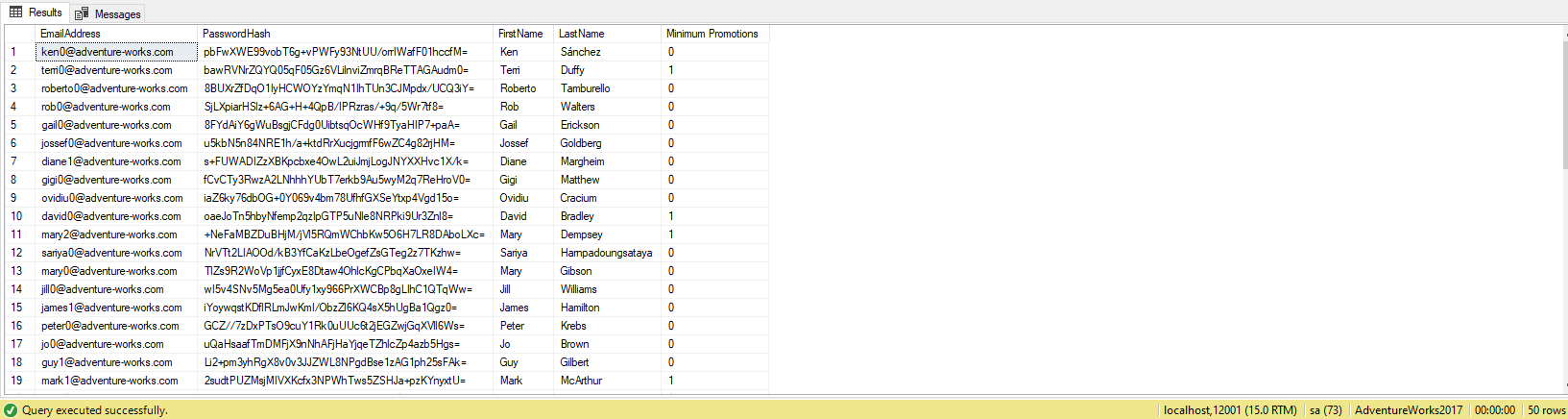
A.EmailAddress,

B.PasswordHash,

C.FirstName,

C.LastName;

--FOR JSON PATH, ROOT('BusinessEntityID'), INCLUDE\_NULL\_VALUES;



**Sample JSON Solution with output (returns 50 objects)**

USE AdventureWorks2017;

DROP FUNCTION IF EXISTS dbo.MinPromotion;

GO

CREATE FUNCTION dbo.MinPromotion

(

@Promotion AS INT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MIN(@Promotion) FROM Person.Person

);

END;

GO

USE AdventureWorks2017;

SELECT TOP 50

A.EmailAddress,

B.PasswordHash,

C.FirstName,

C.LastName,

dbo.MinPromotion(C.EmailPromotion) AS [Minimum Promotions]

FROM Person.EmailAddress AS A

INNER JOIN Person.Password AS B

ON B.BusinessEntityID = A.BusinessEntityID

INNER JOIN Person.Person AS C

ON C.BusinessEntityID = A.BusinessEntityID

WHERE dbo.MinPromotion(C.EmailPromotion) < 2

GROUP BY dbo.MinPromotion(C.EmailPromotion),

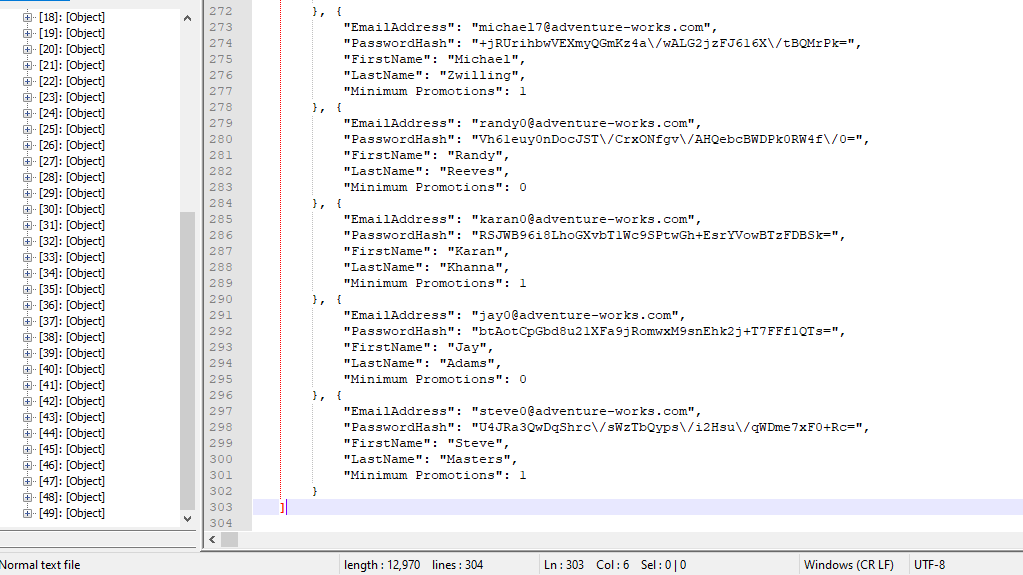
A.EmailAddress,

B.PasswordHash,

C.FirstName,

C.LastName

FOR JSON PATH, ROOT('BusinessEntityID'), INCLUDE\_NULL\_VALUES;



**Complex Query 6**

Scalar function returns the maximum Email promotions. Query returns email address, password hash, first and last name where minimum promotions are greater than or equal to 2

Use AdventureWorks2017 database with Person.EmailAddress, Person.BusinessEntity, and Person.Password tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Person.EmailAddress | EmailAddress | ASC |
| Person.Password | PasswordHash | ASC |
| Person.Person | FirstName  LastName | ASC  ASC |

**Sample relational solution with output (Returns 50 rows)**

USE AdventureWorks2017;

DROP FUNCTION IF EXISTS dbo.MaxPromotion;

GO

CREATE FUNCTION dbo.MaxPromotion

(

@Promotion AS INT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MAX(@Promotion) FROM Person.Person

);

END;

GO

USE AdventureWorks2017;

SELECT TOP 50

A.EmailAddress,

B.PasswordHash,

C.FirstName,

C.LastName,

dbo.MaxPromotion(C.EmailPromotion) AS [Maximum Promotions]

FROM Person.EmailAddress AS A

INNER JOIN Person.Password AS B

ON B.BusinessEntityID = A.BusinessEntityID

INNER JOIN Person.Person AS C

ON C.BusinessEntityID = A.BusinessEntityID

WHERE dbo.MaxPromotion(C.EmailPromotion) >= 2

GROUP BY dbo.MaxPromotion(C.EmailPromotion),

A.EmailAddress,

B.PasswordHash,

C.FirstName,

C.LastName;

--FOR JSON PATH, ROOT('BusinessEntityID'), INCLUDE\_NULL\_VALUES;

**Sample JSON solution with output (Returns 50 objects)**

USE AdventureWorks2017;

DROP FUNCTION IF EXISTS dbo.MaxPromotion;

GO

CREATE FUNCTION dbo.MaxPromotion

(

@Promotion AS INT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MAX(@Promotion) FROM Person.Person

);

END;

GO

USE AdventureWorks2017;

SELECT TOP 50

A.EmailAddress,

B.PasswordHash,

C.FirstName,

C.LastName,

dbo.MaxPromotion(C.EmailPromotion) AS [Maximum Promotions]

FROM Person.EmailAddress AS A

INNER JOIN Person.Password AS B

ON B.BusinessEntityID = A.BusinessEntityID

INNER JOIN Person.Person AS C

ON C.BusinessEntityID = A.BusinessEntityID

WHERE dbo.MaxPromotion(C.EmailPromotion) >= 2

GROUP BY dbo.MaxPromotion(C.EmailPromotion),

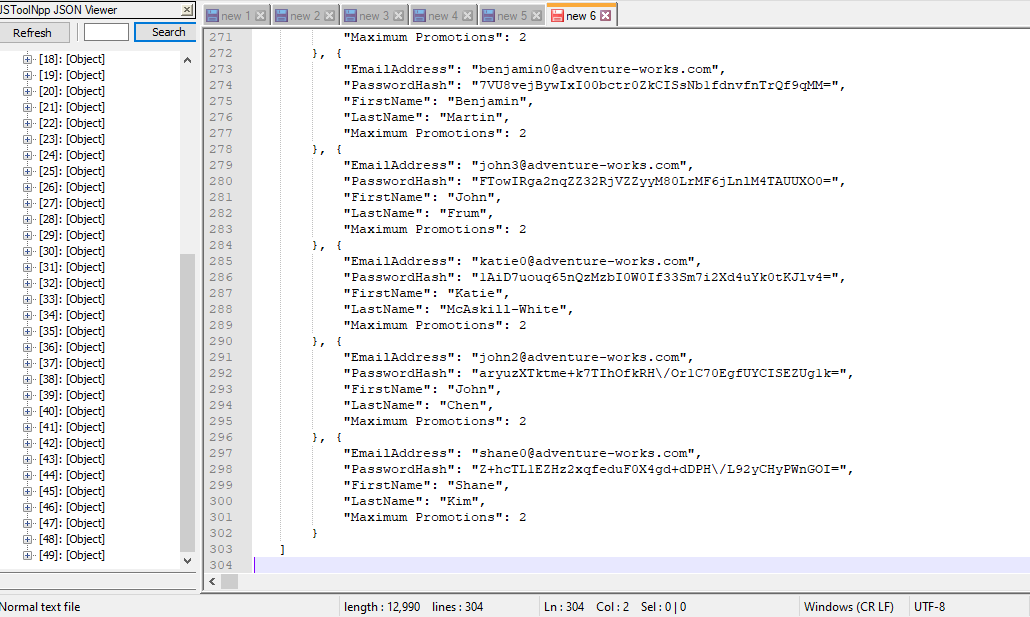
A.EmailAddress,

B.PasswordHash,

C.FirstName,

C.LastName

FOR JSON PATH, ROOT('BusinessEntityID'), INCLUDE\_NULL\_VALUES;



**Complex Query 7**

Scalar function returns the customer id where it was shipped to the USA and their max quantity

Use Northwinds2020TSQLV6 database with Sales.Customer, Sales.Order, Sales.OrderDetail tables

**Standard view**

****

**Key view**

****

**Columns from tables with order by**

|  |  |  |
| --- | --- | --- |
| Table Name | Column Names | Order By |
| Sales.Customer | CustomerID | ASC |
| Sales.[Order] | ShipToCountry | ASC |
| Sales.OrderDetail | Quantity | ASC |

**Sample Relational solution with output (returns 149 rows)**

USE Northwinds2020TSQLV6

DROP FUNCTION IF EXISTS dbo.MAXQuantity;

GO

CREATE FUNCTION dbo.MAXQuantity

(

@quantity AS SMALLINT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MAX(@quantity) FROM Sales.[ORDERDetail]

);

END;

GO

USE Northwinds2020TSQLV6;

SELECT A.CustomerId,

B.ShipToCountry,

dbo.MAXQuantity(C.Quantity) AS [Maximum Quantity]

FROM Sales.Customer AS A

INNER JOIN Sales.[Order] AS B

ON B.CustomerId = A.CustomerId

INNER JOIN Sales.OrderDetail AS C

ON C.OrderId = B.OrderId

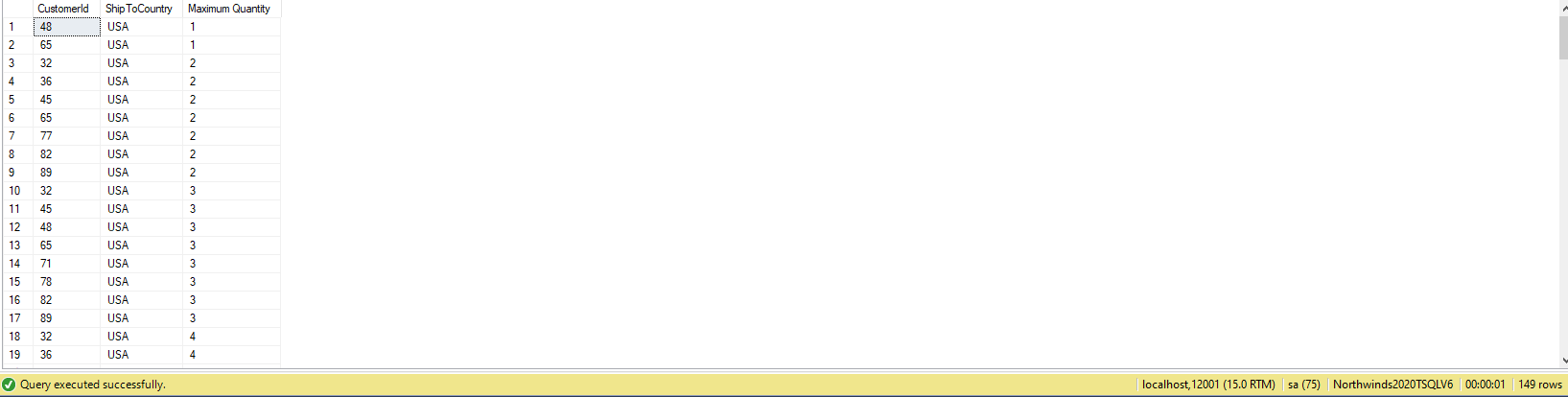
WHERE B.ShipToCountry LIKE 'USA'

GROUP BY dbo.MAXQuantity(C.Quantity),

A.CustomerId,

B.ShipToCountry;

--FOR JSON PATH, ROOT('CustomerId'), INCLUDE\_NULL\_VALUES;



**Sample JSON solution with output (Returns Objects 149)**

USE Northwinds2020TSQLV6

DROP FUNCTION IF EXISTS dbo.MAXQuantity;

GO

CREATE FUNCTION dbo.MAXQuantity

(

@quantity AS SMALLINT

)

RETURNS SMALLINT

AS

BEGIN

RETURN

(

SELECT MAX(@quantity) FROM Sales.[ORDERDetail]

);

END;

GO

USE Northwinds2020TSQLV6;

SELECT A.CustomerId,

B.ShipToCountry,

dbo.MAXQuantity(C.Quantity) AS [Maximum Quantity]

FROM Sales.Customer AS A

INNER JOIN Sales.[Order] AS B

ON B.CustomerId = A.CustomerId

INNER JOIN Sales.OrderDetail AS C

ON C.OrderId = B.OrderId

WHERE B.ShipToCountry LIKE 'USA'

GROUP BY dbo.MAXQuantity(C.Quantity),

A.CustomerId,

B.ShipToCountry

FOR JSON PATH, ROOT('CustomerId'), INCLUDE\_NULL\_VALUES;

