Work with windowing functions and other aggregations

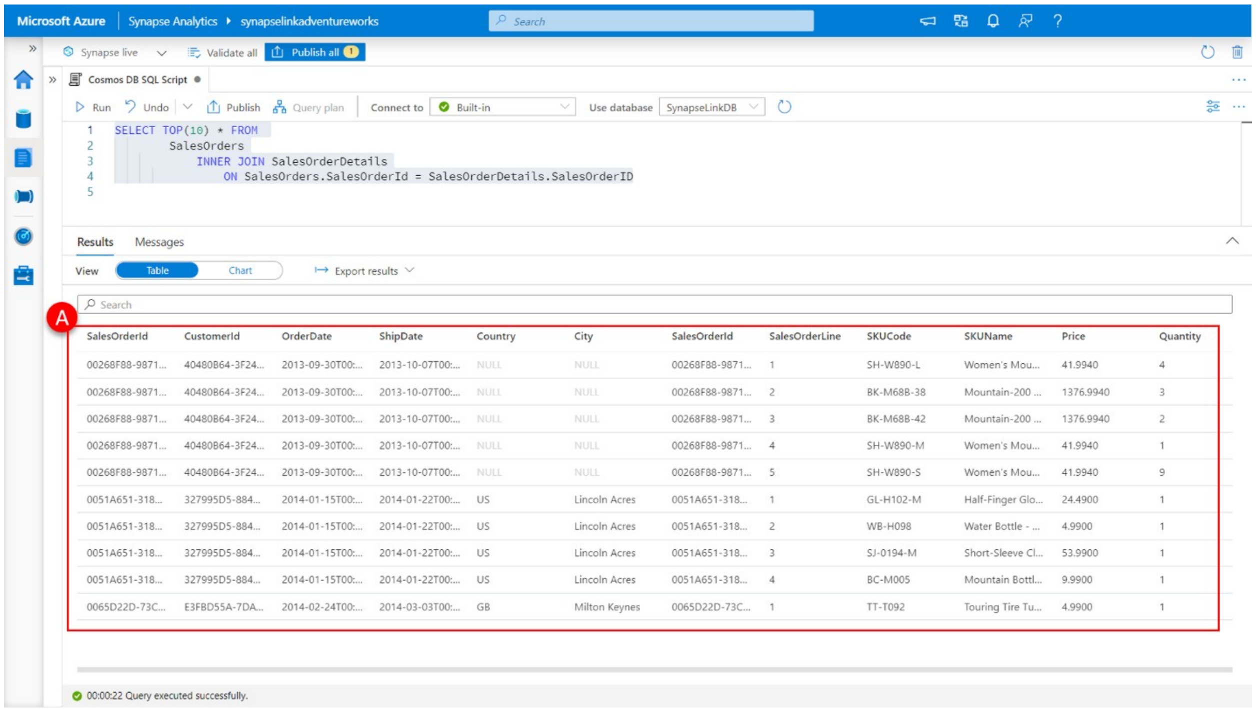
Note In this reading you can see the steps involved in working with windowing functions and other aggregations.

Adventure Works wanted to be able to understand how the sales order volume and revenue is distributed by city, for those customers where they have address details. In the previous units, we prepared a SalesOrderView that contains a row for every customer sales order, with the country and city information for that customer, where that information was available, and a SalesOrderDetailsView that contains a row for every sale order line, with information on the price and quantity and details of the product sold.

If we join the SalesOrders and SalesOrderDetails views, we created in the previous units, using the SalesOrderId of both we will get a **result set (A)**, where a single row has both the dimensions across which we want to summarize this data, along with the values that we wish to measure.

Paste the following SQL into the query pane.

```
SELECT TOP(10) * FROM
        SalesOrders
            INNER JOIN SalesOrderDetails
                ON SalesOrders.SalesOrderId = SalesOrderDetails.SalesOrderID
```



```sql
SELECT TOP(10) * FROM
        SalesOrders
            INNER JOIN SalesOrderDetails
                ON SalesOrders.SalesOrderId = SalesOrderDetails.SalesOrderID
```

SalesOrderId	CustomerId	OrderDate	ShipDate	Country	City	SalesOrderId	SalesOrderLine	SKUCode	SKUName	Price	Quantity
00268F88-9871...	40480B64-3F24...	2013-09-30T00:...	2013-10-07T00:...	NULL	NULL	00268F88-9871...	1	SH-W890-L	Women's Mou...	41.9940	4
00268F88-9871...	40480B64-3F24...	2013-09-30T00:...	2013-10-07T00:...	NULL	NULL	00268F88-9871...	2	BK-M68B-38	Mountain-200 ...	1376.9940	3
00268F88-9871...	40480B64-3F24...	2013-09-30T00:...	2013-10-07T00:...	NULL	NULL	00268F88-9871...	3	BK-M68B-42	Mountain-200 ...	1376.9940	2
00268F88-9871...	40480B64-3F24...	2013-09-30T00:...	2013-10-07T00:...	NULL	NULL	00268F88-9871...	4	SH-W890-M	Women's Mou...	41.9940	1
00268F88-9871...	40480B64-3F24...	2013-09-30T00:...	2013-10-07T00:...	NULL	NULL	00268F88-9871...	5	SH-W890-S	Women's Mou...	41.9940	9
0051A651-318...	327995D5-884...	2014-01-15T00:...	2014-01-22T00:...	US	Lincoln Acres	0051A651-318...	1	GL-H102-M	Half-Finger Glo...	24.4900	1
0051A651-318...	327995D5-884...	2014-01-15T00:...	2014-01-22T00:...	US	Lincoln Acres	0051A651-318...	2	WB-H098	Water Bottle - ...	4.9900	1
0051A651-318...	327995D5-884...	2014-01-15T00:...	2014-01-22T00:...	US	Lincoln Acres	0051A651-318...	3	SJ-0194-M	Short-Sleeve Cl...	53.9900	1
0051A651-318...	327995D5-884...	2014-01-15T00:...	2014-01-22T00:...	US	Lincoln Acres	0051A651-318...	4	BC-M005	Mountain Bottl...	9.9900	1
0065D22D-73C...	E3FBD55A-7DA...	2014-02-24T00:...	2014-03-03T00:...	GB	Milton Keynes	0065D22D-73C...	1	TT-T092	Touring Tire Tu...	4.9900	1

00:00:22 Query executed successfully.

Querying joined views in Azure Synapse Studio

Click **run**.

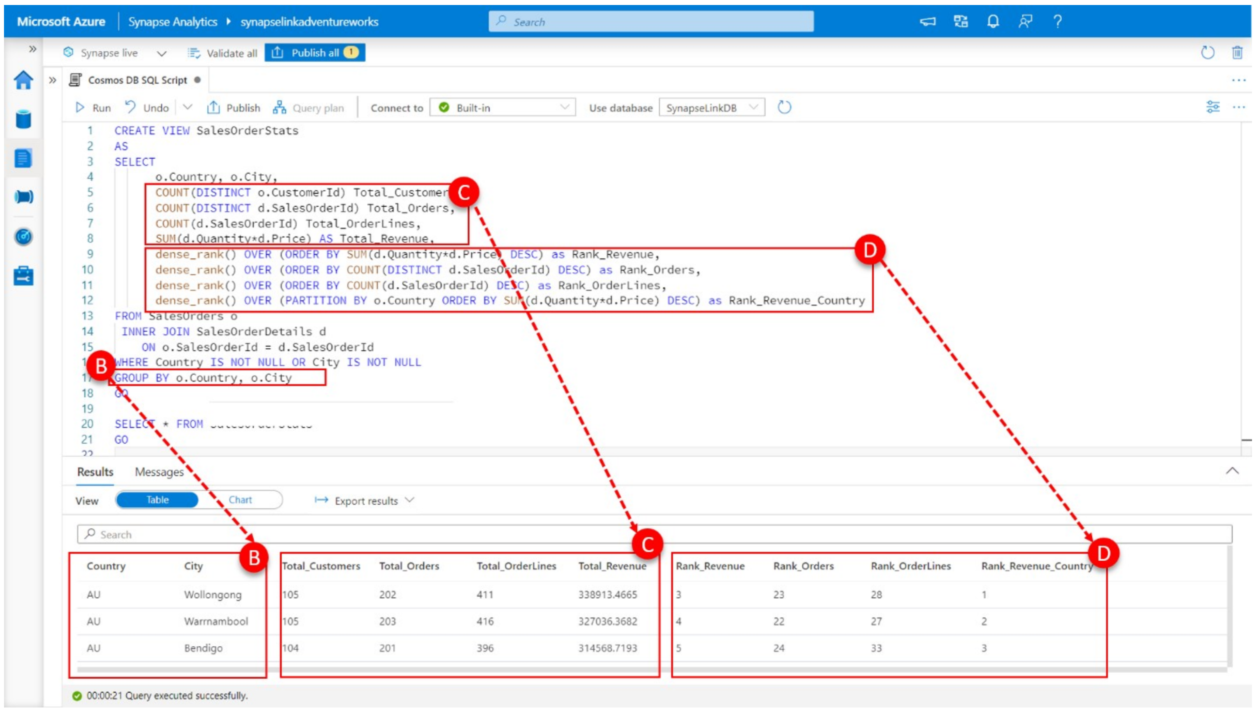
We can now create a single view, that will provide us with Country and City level statistics we are looking for.

Paste the following SQL into the query pane.

```
CREATE VIEW SalesOrderStats
AS
SELECT
      o.Country, o.City,
      COUNT(DISTINCT o.CustomerId) Total_Customers,
      COUNT(DISTINCT d.SalesOrderId) Total_Orders,
      COUNT(d.SalesOrderId) Total_OrderLines,
      SUM(d.Quantity*d.Price) AS Total_Revenue,
      dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
      dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_O
rders,
      dense_rank() OVER (ORDER BY COUNT(d.SalesOrderId) DESC) as Rank_OrderLines
,
      dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price)
DESC) as Rank_Revenue_Country
FROM SalesOrders o
 INNER JOIN SalesOrderDetails d
    ON o.SalesOrderId = d.SalesOrderId
WHERE Country IS NOT NULL OR City IS NOT NULL
GROUP BY o.Country, o.City
GO

SELECT * FROM SalesOrderStats
GO
```

The query above will create the SalesOrderStats view and output the results of the view created.



```sql
CREATE VIEW SalesOrderStats
AS
SELECT
        o.Country, o.City,
        COUNT(DISTINCT o.CustomerId) Total_Customer
        COUNT(DISTINCT d.SalesOrderId) Total_Orders,
        COUNT(d.SalesOrderId) Total_OrderLines,
        SUM(d.Quantity*d.Price) AS Total_Revenue,
        dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
        dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
        dense_rank() OVER (ORDER BY COUNT(d.SalesOrderId) DESC) as Rank_OrderLines,
        dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
FROM SalesOrders o
 INNER JOIN SalesOrderDetails d
    ON o.SalesOrderId = d.SalesOrderId
WHERE Country IS NOT NULL OR City IS NOT NULL
GROUP BY o.Country, o.City
GO

SELECT * FROM SalesOrderStats
GO
```

Country	City	Total_Customers	Total_Orders	Total_OrderLines	Total_Revenue	Rank_Revenue	Rank_Orders	Rank_OrderLines	Rank_Revenue_Country
AU	Wollongong	105	202	411	338913.4665	3	23	28	1
AU	Warrnambool	105	203	416	327036.3682	4	22	27	2
AU	Bendigo	104	201	396	314568.7193	5	24	33	3

00:00:21 Query executed successfully.

Creating and querying a view in Azure Synapse Studio
Click **run**.

The query captured in this view answers many parts of the questions being asked through traditional aggregation, because we are mostly interested in understanding the number (COUNT) or total (SUM) of values, a GROUP BY clause that covers both **Country and City (B)** can answer most of the questions, with absolute values for the **total number of customers, orders and order lines and the sum of revenue by City (C)**.

To answer the ranking part of the question, we use window functions. In essence, a window function calculates a result for every row of a table based on a group of rows, called the frame. Every row can have a unique frame associated with it for that window function, allowing you to concisely express and solve ranking, analytic, and aggregation problems in a powerful yet simple manner that no other approach provides.

Here we **use the dense_rank() function to calculate the rank of each city by revenue, number of orders and total order lines (D)**.

As well as the rank of each city within each country, by partitioning the dense_rank() window function by the country.