

Work with the windowing function and other aggregations

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

Adventure Works wants 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 associated with the product sold. Together these views contain all the raw data we need to answer these questions:

1. Paste the code below into a **new cell (A)**, and click the **run cell** button.

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
%%sql
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_Ord
ers,
       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) D
ESC) as Rank_Revenue_Country
```

```

FROM SalesOrderView o
INNER JOIN SalesOrderDetailsView d
    ON o.SalesOrderId = d.SalesOrderId
WHERE Country IS NOT NULL OR City IS NOT NULL
GROUP BY o.Country, o.City
ORDER BY Total_Revenue DESC
LIMIT 10

```

The screenshot shows a notebook interface with a SparkSQL query and its execution results. Red annotations highlight key parts of the query and the output table:

- A**: Points to the `Rank_Revenue` column in the query and the `Total_Revenue` column in the results table.
- B**: Points to the `GROUP BY o.Country, o.City` clause in the query and the first two columns (`Country`, `City`) in the results table.
- C**: Points to the `ORDER BY Total_Revenue DESC` clause in the query and the `Total_Revenue` column in the results table.

Country	City	Total_Customers	Total_Orders	Total_OrderLines	Total_Revenue
GB	London	420	686	1579	802810.2968999999
FR	Paris	386	522	1174	539725.7999999999
AU	Wollongong	105	202	411	338913.4665
AU	Warrnambool	105	203	416	327036.36819999997
AU	Bendigo	104	201	396	314568.7192999999

Using SparkSQL to create a windowing function in a notebook

This query answers the questions being asked through traditional aggregation, as 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 to calculate 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 does.

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.

Microsoft Azure | Synapse Analytics | synapseinkadventureworks

Synapse live | Validate all | Publish all

Cosmos DB Notebook

Attach to: adventurespark | Language: PySpark (Python) | Preview Features

```

1 %sql
2 SELECT o.Country, o.City,
3        COUNT(DISTINCT o.CustomerId) Total_Customers,
4        COUNT(DISTINCT d.SalesOrderId) Total_Orders,
5        COUNT(d.SalesOrderId) Total_OrderLines,
6        SUM(d.Quantity*d.Price) AS Total_Revenue,
7        dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
8        dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
9        dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_OrderLines,
10       dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
11 FROM SalesOrderView o
12 INNER JOIN SalesOrderDetailsView d
13 ON o.SalesOrderId = d.SalesOrderId
14 WHERE Country IS NOT NULL OR City IS NOT NULL
15 GROUP BY o.Country, o.City
16 ORDER BY Total_Revenue DESC
17 LIMIT 10
18

```

Command executed in 10s 154ms

Job execution Succeeded Spark 2 executors 16 cores

View in monitoring | Open Spark UI

Total_OrderLines	Total_Revenue	Rank_Revenue	Rank_Orders	Rank_OrderLines	Rank_Revenue_Country
1579	802810.2968999991	1	1	1	1
1174	539725.7999999999	2	2	2	1
411	338913.4665	3	23	28	1
416	327036.36819999997	4	22	27	2
396	314568.7192999999	5	24	33	3

Using the dense_rank windowing function in a notebook

Given the usefulness of these results we can encapsulate them in a temporary view for future use:

2. Paste the code below into a **new cell (E)**, and click the **run cell** button.

```

CREATE OR REPLACE TEMPORARY VIEW SalesOrderStatsView
AS
SELECT o.Country, o.City,
       COUNT(DISTINCT o.CustomerId) Total_Customers,
       COUNT(DISTINCT d.SalesOrderId) Total_Orders,

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

```

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_Ord
ers,
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) D
ESC) as Rank_Revenue_Country
FROM SalesOrderView o
INNER JOIN SalesOrderDetailsView d
    ON o.SalesOrderId = d.SalesOrderId
WHERE Country IS NOT NULL OR City IS NOT NULL
GROUP BY o.Country, o.City
ORDER BY Total_Revenue DESC

```

The screenshot shows the Microsoft Azure Synapse Analytics interface. At the top, there's a search bar and navigation icons. Below that, the 'Cosmos DB Notebook' tab is active. The notebook contains a SQL query that creates a temporary view named 'SalesOrderStatsView' and selects various aggregated metrics from 'SalesOrderView' and 'SalesOrderDetailsView'. The query includes counts for customers, orders, and order lines, as well as revenue calculations and dense ranking. The notebook interface includes a 'Run all' button and a 'Publish' button. Below the query, the execution results are displayed, showing a message 'Command executed in 2s 33ms' and a 'View' button with 'Table' and 'Chart' options. The results area currently shows 'No Data Available!'.

```

1 --sql
2 CREATE OR REPLACE TEMPORARY VIEW SalesOrderStatsView
3 AS
4 SELECT o.Country, o.City,
5        COUNT(DISTINCT o.CustomerId) Total_Customers,
6        COUNT(DISTINCT d.SalesOrderId) Total_Orders,
7        COUNT(d.SalesOrderId) Total_OrderLines,
8        SUM(d.Quantity*d.Price) AS Total_Revenue,
9        dense_rank() OVER (ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue,
10       dense_rank() OVER (ORDER BY COUNT(DISTINCT d.SalesOrderId) DESC) as Rank_Orders,
11       dense_rank() OVER (ORDER BY COUNT(d.SalesOrderId) DESC) as Rank_OrderLines,
12       dense_rank() OVER (PARTITION BY o.Country ORDER BY SUM(d.Quantity*d.Price) DESC) as Rank_Revenue_Country
13 FROM SalesOrderView o
14 INNER JOIN SalesOrderDetailsView d
15     ON o.SalesOrderId = d.SalesOrderId
16 WHERE Country IS NOT NULL OR City IS NOT NULL
17 GROUP BY o.Country, o.City
18 ORDER BY Total_Revenue DESC
19

```

Command executed in 2s 33ms

View

No Data Available!