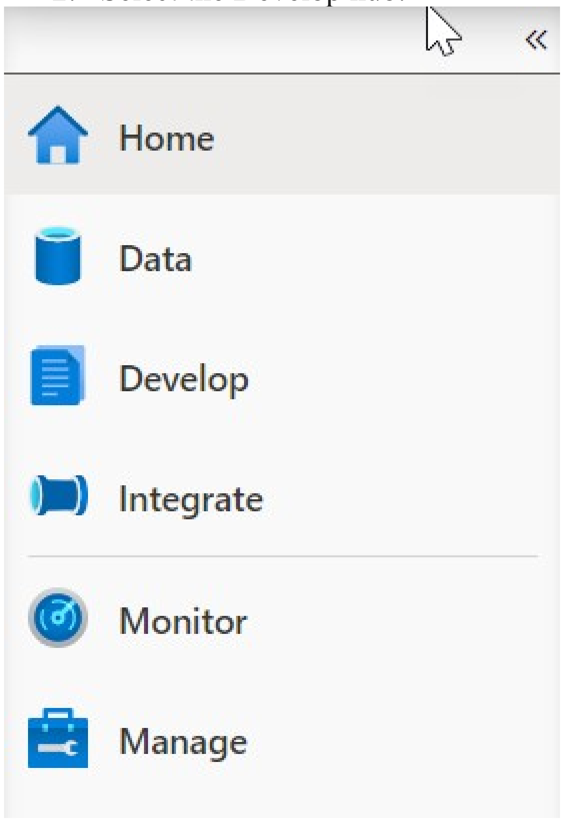
# Work with windowing functions

**Note:**

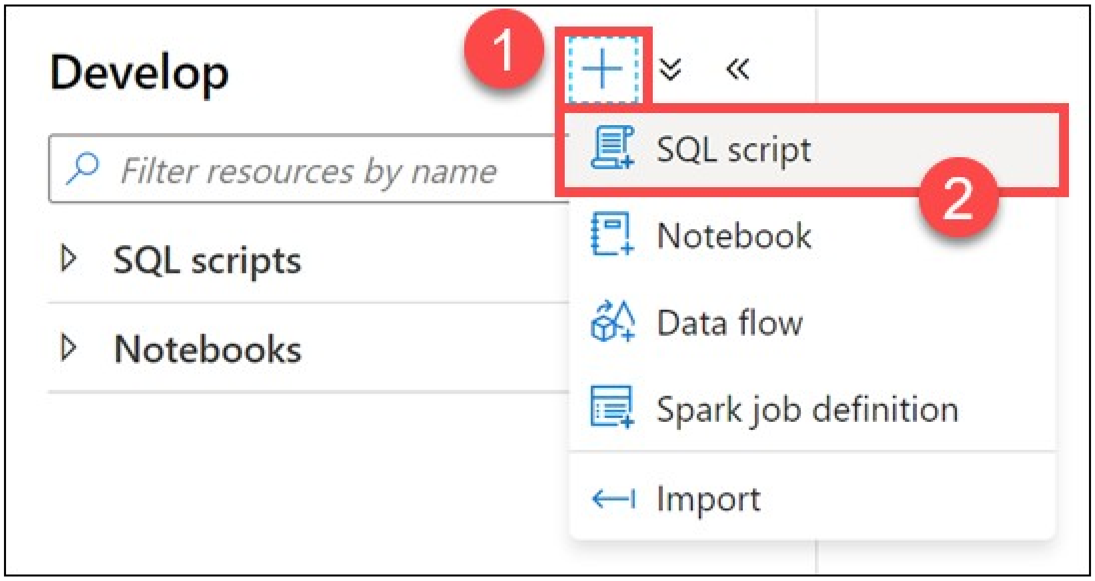
You are not required to complete the processes, tasks, activities, or steps presented in this example. The various samples provided are for illustrative purposes only and it's likely that if you try this out you will encounter issues in your system.

One of the key components of window functions is the OVER clause. This clause determines the partitioning and ordering of a rowset before the associated window function is applied. That is, the OVER clause defines a window or user-specified set of rows within a query result set. A window function then computes a value for each row in the window. You can use the OVER clause with functions to compute aggregated values such as moving averages, cumulative aggregates, running totals, or a top N per group results.

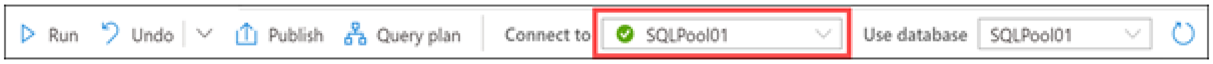
1. Open Synapse Studio (https://web.azuresynapse.net/).
2. Select the Develop hub.



3. From the Develop menu, select the + button (1) and choose SQL Script (2) from the context menu.



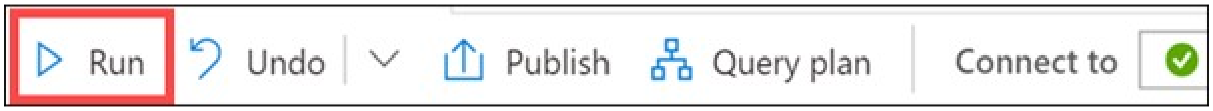
4. In the toolbar menu, connect to the SQL Pool database to execute the query.



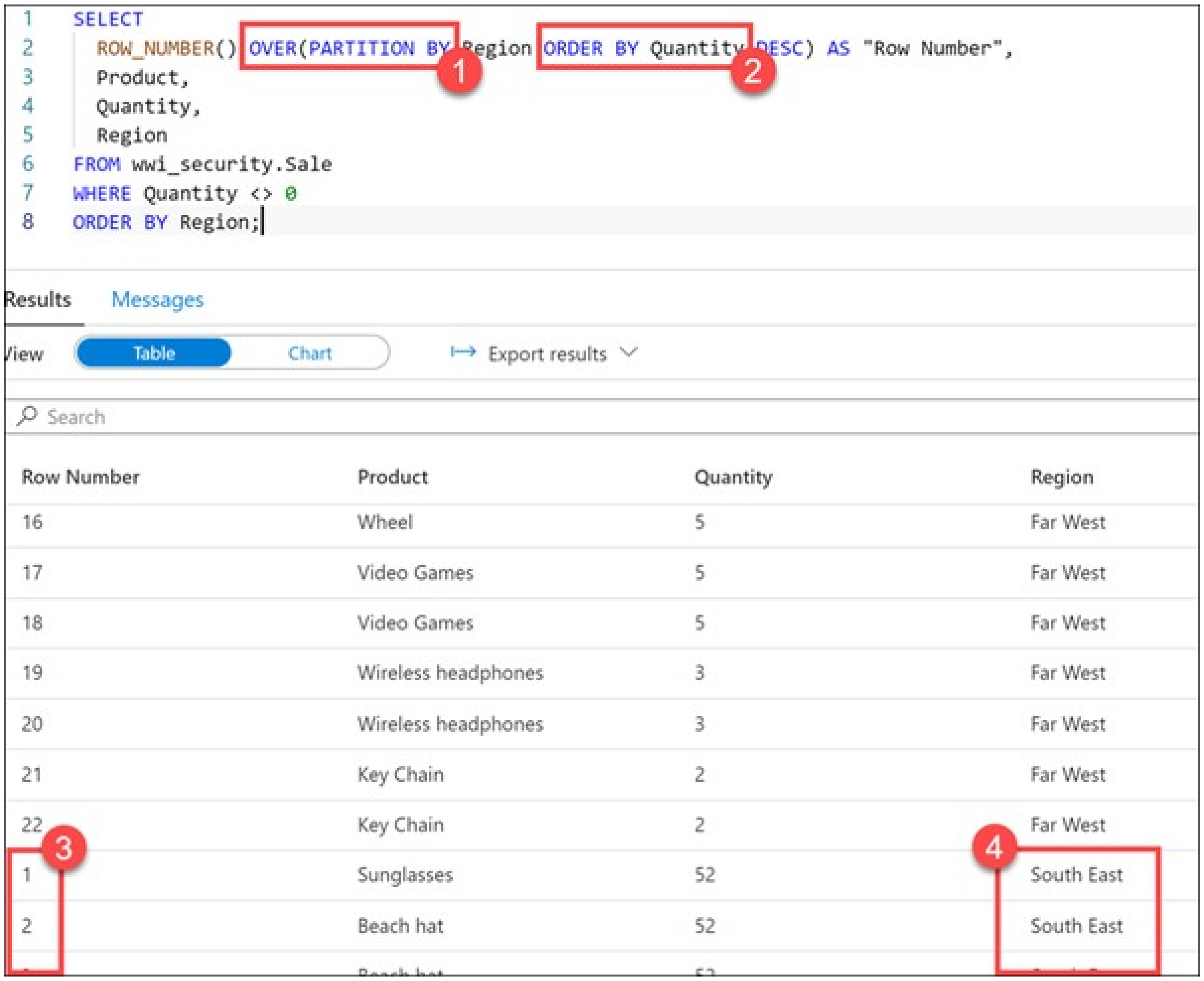
5. In the query window, replace the script with the following to use the OVER clause with data from the wwi_security.Sale table:

```sql
SELECT
ROW_NUMBER() OVER(PARTITION BY Region ORDER BY Quantity DESC) AS "Row Number",
Product,
Quantity,
Region
FROM wwi_security.Sale
WHERE Quantity <> 0
ORDER BY Region;
```

6. Select Run from the toolbar menu to execute the SQL command.



When we use PARTITION BY with the OVER clause (1), we divide the query result set into partitions. The window function is applied to each partition separately and computation restarts for each partition.



```
SELECT
    ROW_NUMBER() OVER(PARTITION BY Region ORDER BY Quantity DESC) AS "Row Number",
    Product,
    Quantity,
    Region
FROM wwi_security.Sale
WHERE Quantity <> 0
ORDER BY Region;
```

| Row Number | Product | Quantity | Region |
|---|---|---|---|
| 16 | Wheel | 5 | Far West |
| 17 | Video Games | 5 | Far West |
| 18 | Video Games | 5 | Far West |
| 19 | Wireless headphones | 3 | Far West |
| 20 | Wireless headphones | 3 | Far West |
| 21 | Key Chain | 2 | Far West |
| 22 | Key Chain | 2 | Far West |
| 1 | Sunglasses | 52 | South East |
| 2 | Beach hat | 52 | South East |

The script we executed uses the OVER clause with ROW_NUMBER function (1) to display a row number for each row within a partition. The partition in our case is the Region column. The ORDER BY clause (2) specified in the OVER clause orders the rows in each partition by the column Quantity. The ORDER BY clause in the SELECT statement determines the order in which the entire query result set is returned.

Scroll down in the results view until the Row Number count (3) starts over with a different region (4). Since the partition is set to Region, the ROW_NUMBER resets when the region changes. Essentially, we have partitioned by region and have a result set identified by the number of rows in that region.

## Aggregate functions

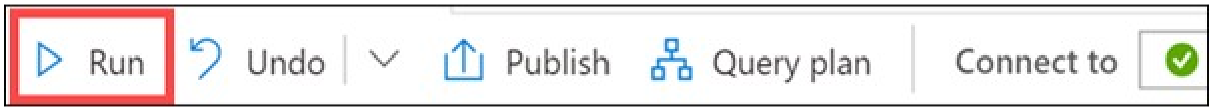
Now use aggregate functions with our window by expanding on our query that uses the OVER clause.

1. In the query window, replace the script with the following to add aggregate functions:

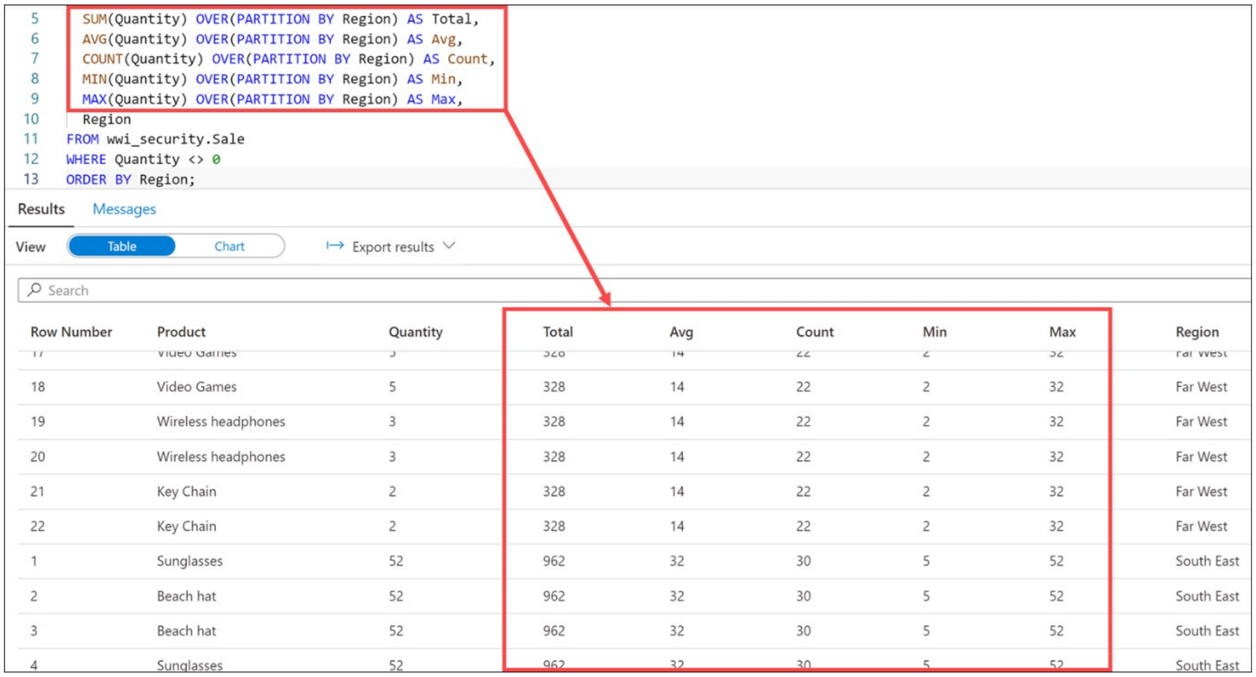
```
1
2
```

```sql
SELECT
ROW_NUMBER() OVER(PARTITION BY Region ORDER BY Quantity DESC) AS "Row Number",
Product,
Quantity,
SUM(Quantity) OVER(PARTITION BY Region) AS Total,
AVG(Quantity) OVER(PARTITION BY Region) AS Avg,
COUNT(Quantity) OVER(PARTITION BY Region) AS Count,
MIN(Quantity) OVER(PARTITION BY Region) AS Min,
MAX(Quantity) OVER(PARTITION BY Region) AS Max,
Region
FROM wwi_security.Sale
WHERE Quantity <> 0
ORDER BY Region;
```

2. Select Run from the toolbar menu to execute the SQL command.



In our query, we added the SUM, AVG, COUNT, MIN, and MAX aggregate functions. Using the OVER clause is more efficient than using subqueries.



```sql
    SUM(Quantity) OVER(PARTITION BY Region) AS Total,
    AVG(Quantity) OVER(PARTITION BY Region) AS Avg,
    COUNT(Quantity) OVER(PARTITION BY Region) AS Count,
    MIN(Quantity) OVER(PARTITION BY Region) AS Min,
    MAX(Quantity) OVER(PARTITION BY Region) AS Max,
    Region
FROM wwi_security.Sale
WHERE Quantity <> 0
ORDER BY Region;
```

| Row Number | Product | Quantity | Total | Avg | Count | Min | Max | Region |
|---|---|---|---|---|---|---|---|---|
| 17 | Video Games | 5 | 328 | 14 | 22 | 2 | 32 | Far West |
| 18 | Video Games | 5 | 328 | 14 | 22 | 2 | 32 | Far West |
| 19 | Wireless headphones | 3 | 328 | 14 | 22 | 2 | 32 | Far West |
| 20 | Wireless headphones | 3 | 328 | 14 | 22 | 2 | 32 | Far West |
| 21 | Key Chain | 2 | 328 | 14 | 22 | 2 | 32 | Far West |
| 22 | Key Chain | 2 | 328 | 14 | 22 | 2 | 32 | Far West |
| 1 | Sunglasses | 52 | 962 | 32 | 30 | 5 | 52 | South East |
| 2 | Beach hat | 52 | 962 | 32 | 30 | 5 | 52 | South East |
| 3 | Beach hat | 52 | 962 | 32 | 30 | 5 | 52 | South East |
| 4 | Sunglasses | 52 | 962 | 32 | 30 | 5 | 52 | South East |

## Analytic functions

Analytic functions calculate an aggregate value based on a group of rows. Unlike aggregate functions, however, analytic functions can return multiple rows for each group. Use analytic functions to compute moving averages, running totals, percentages, or top-N results within a group.

Tailwind Traders has book sales data they import from their online store and wish to compute percentages of book downloads by category.

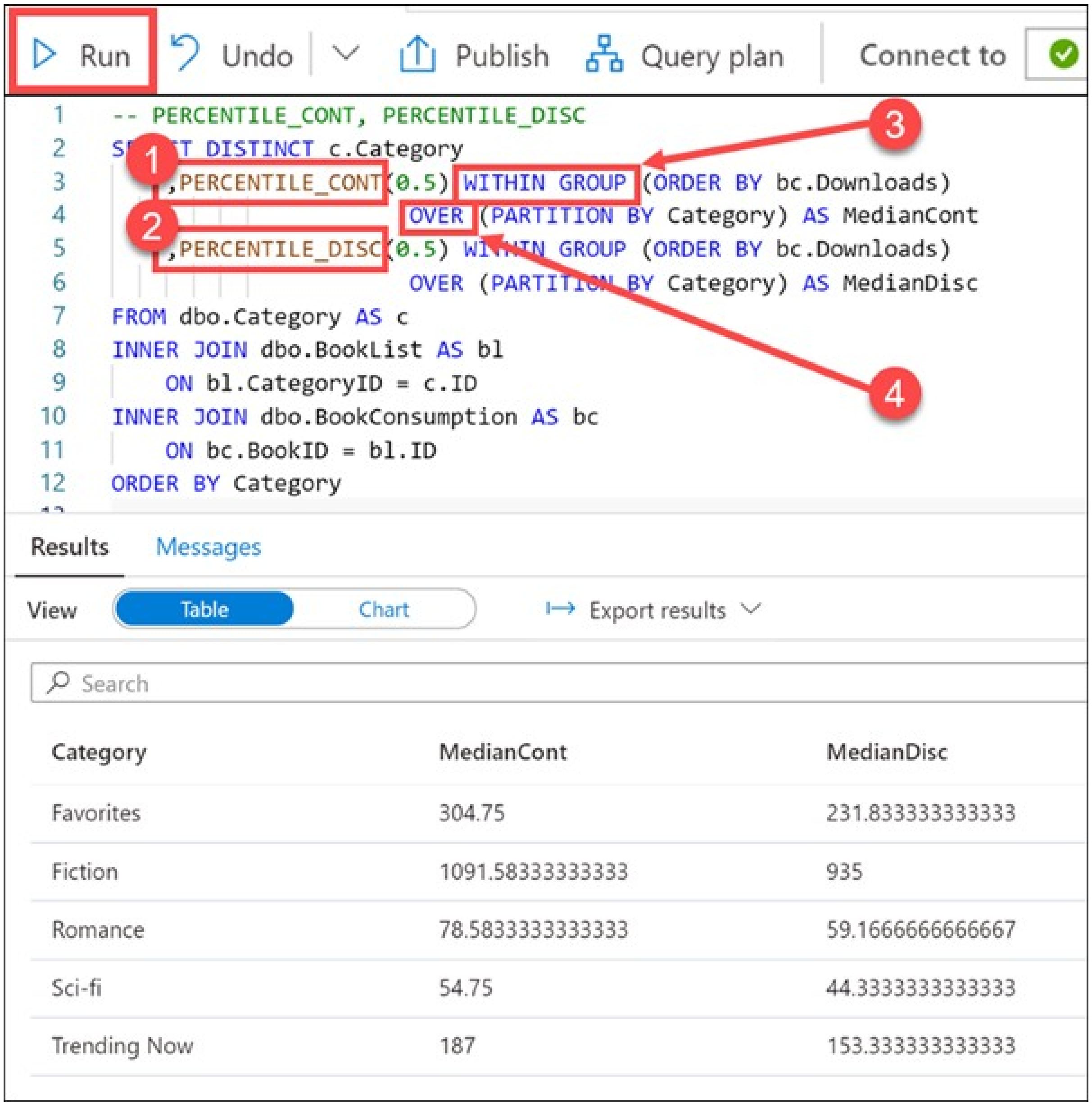
To do this, you decide to build window functions that use the PERCENTILE_CONT and PERCENTILE_DISC functions.

1. In the query window, replace the script with the following to add aggregate functions:

```sql
-- PERCENTILE_CONT, PERCENTILE_DISC
SELECT DISTINCT c.Category
```

```
,PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY bc.Downloads)
                    OVER (PARTITION BY Category) AS MedianCont
,PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY bc.Downloads)
                    OVER (PARTITION BY Category) AS MedianDisc
FROM dbo.Category AS c
INNER JOIN dbo.BookList AS bl
    ON bl.CategoryID = c.ID
INNER JOIN dbo.BookConsumption AS bc
    ON bc.BookID = bl.ID
ORDER BY Category
```

2. Select Run from the toolbar menu to execute the SQL command.



```
-- PERCENTILE_CONT, PERCENTILE_DISC
SELECT DISTINCT c.Category
    ,PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY bc.Downloads)
                    OVER (PARTITION BY Category) AS MedianCont
    ,PERCENTILE_DISC(0.5) WITHIN GROUP (ORDER BY bc.Downloads)
                    OVER (PARTITION BY Category) AS MedianDisc
FROM dbo.Category AS c
INNER JOIN dbo.BookList AS bl
    ON bl.CategoryID = c.ID
INNER JOIN dbo.BookConsumption AS bc
    ON bc.BookID = bl.ID
ORDER BY Category
```

| Category | MedianCont | MedianDisc |
|---|---|---|
| Favorites | 304.75 | 231.833333333333 |
| Fiction | 1091.58333333333 | 935 |
| Romance | 78.5833333333333 | 59.1666666666667 |
| Sci-fi | 54.75 | 44.3333333333333 |
| Trending Now | 187 | 153.333333333333 |

In this query, we use PERCENTILE_CONT (1) and PERCENTILE_DISC (2) to find the median number of downloads in each book category. These functions may not return the same value. PERCENTILE_CONT interpolates the appropriate value, which may or may not exist in the data set, while PERCENTILE_DISC always returns an actual value from the set. To explain further, PERCENTILE_DISC computes a specific percentile for sorted values in an entire rowset or within a rowset's distinct partitions.

The 0.5 value passed to the percentile functions (1 & 2) computes the 50th percentile, or the median, of the downloads.

The WITHIN GROUP expression (3) specifies a list of values to sort and compute the percentile over. Only one ORDER BY expression is allowed, and the default sort order is ascending.

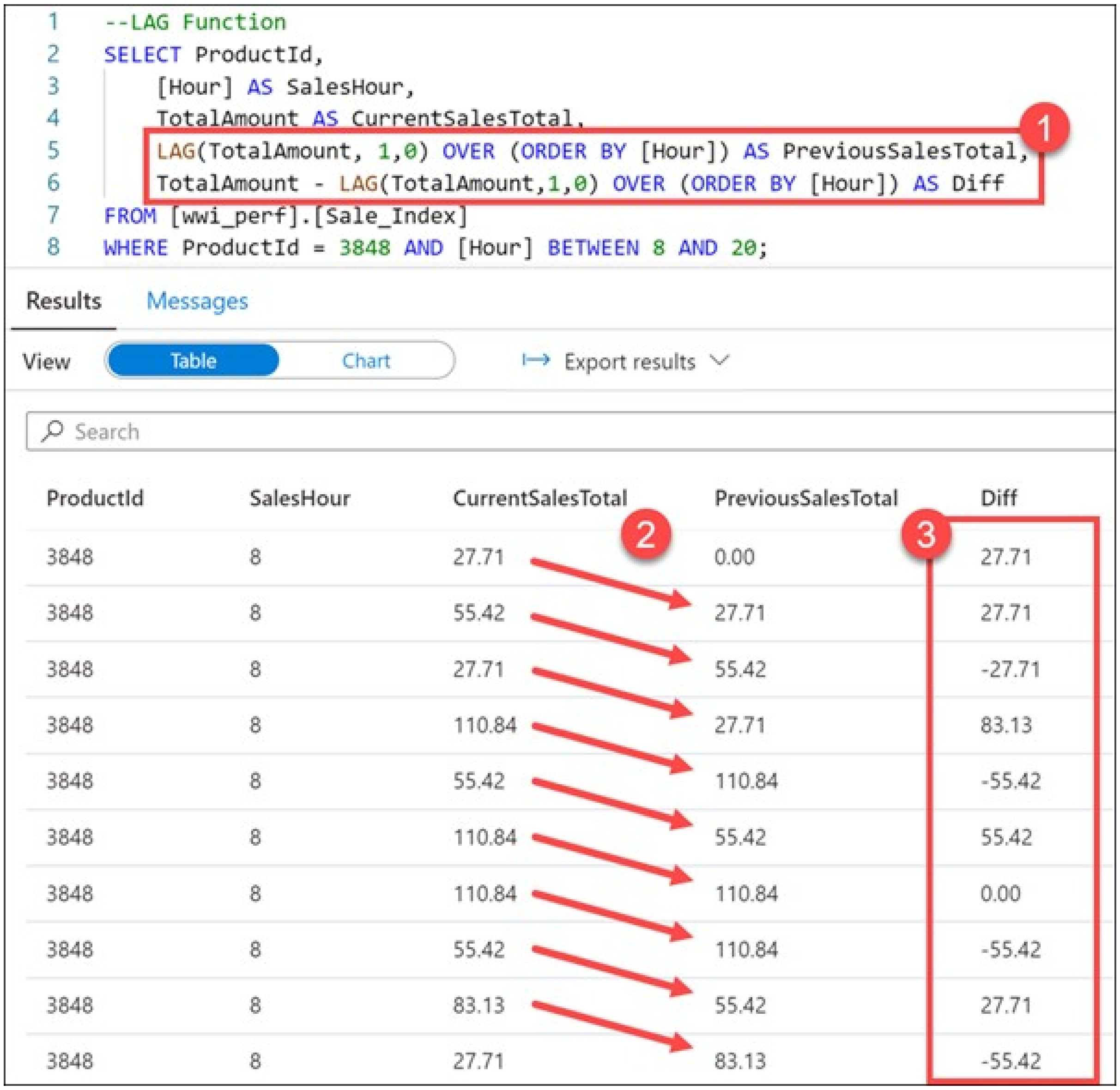
The OVER clause (4) divides the FROM clause's result set into partitions, in this case, Category. The percentile function is applied to these partitions.

3. In the query window, replace the script with the following to use the LAG analytic function:

```sql
--LAG Function
SELECT ProductId,
    [Hour] AS SalesHour,
    TotalAmount AS CurrentSalesTotal,
    LAG(TotalAmount, 1,0) OVER (ORDER BY [Hour]) AS PreviousSalesTotal,
    TotalAmount - LAG(TotalAmount,1,0) OVER (ORDER BY [Hour]) AS Diff
FROM [wwi_perf].[Sale_Index]
WHERE ProductId = 3848 AND [Hour] BETWEEN 8 AND 20;
```

Tailwind Traders wants to compare the sales totals for a product over an hourly basis over time, showing the difference in value.

To accomplish this, you use the LAG analytic function. This function accesses data from a previous row in the same result set without the use of a self-join. LAG provides access to a row at a given physical offset that comes before the current row. We use this analytic function to compare values in the current row with values in a previous row.



```sql
--LAG Function
SELECT ProductId,
    [Hour] AS SalesHour,
    TotalAmount AS CurrentSalesTotal,
    LAG(TotalAmount, 1,0) OVER (ORDER BY [Hour]) AS PreviousSalesTotal,
    TotalAmount - LAG(TotalAmount,1,0) OVER (ORDER BY [Hour]) AS Diff
FROM [wwi_perf].[Sale_Index]
WHERE ProductId = 3848 AND [Hour] BETWEEN 8 AND 20;
```

| ProductId | SalesHour | CurrentSalesTotal | PreviousSalesTotal | Diff |
|---|---|---|---|---|
| 3848 | 8 | 27.71 | 0.00 | 27.71 |
| 3848 | 8 | 55.42 | 27.71 | 27.71 |
| 3848 | 8 | 27.71 | 55.42 | -27.71 |
| 3848 | 8 | 110.84 | 27.71 | 83.13 |
| 3848 | 8 | 55.42 | 110.84 | -55.42 |
| 3848 | 8 | 110.84 | 55.42 | 55.42 |
| 3848 | 8 | 110.84 | 110.84 | 0.00 |
| 3848 | 8 | 55.42 | 110.84 | -55.42 |
| 3848 | 8 | 83.13 | 55.42 | 27.71 |
| 3848 | 8 | 27.71 | 83.13 | -55.42 |

In this query, we use the LAG function (1) to return the difference in sales (2) for a specific product over peak sales hours (8-20). We also calculate the difference in sales from one row to the next (3). Notice that because there is no lag value available for the first row, the default of zero (0) is returned.

## ROWS clause

The ROWS and RANGE clauses further limit the rows within the partition by specifying start and end points within the partition. This is done by specifying a range of rows with respect to the current row either by logical association or physical association. Physical association is achieved by using the ROWS clause.

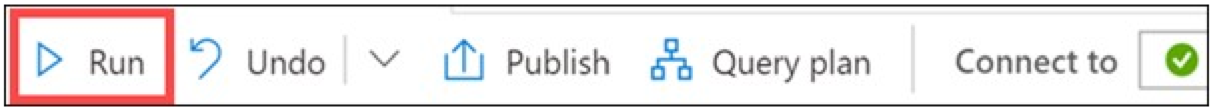
Tailwind Traders wants to find the books that have the fewest downloads by country, while displaying the total number of downloads for each book within each country in ascending order.

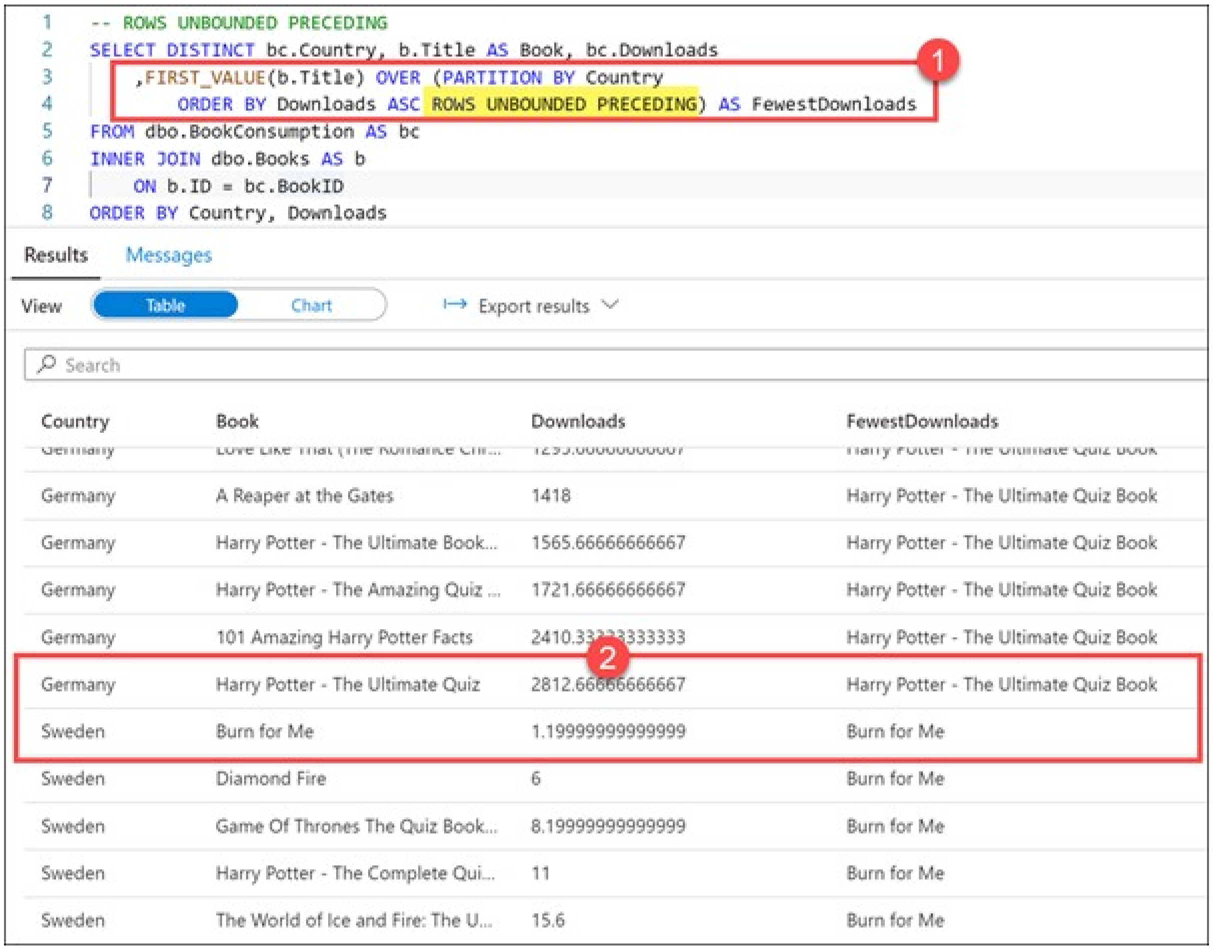
To achieve this, you use ROWS in combination with UNBOUNDED PRECEDING to limit the rows within the Country partition, specifying that the window start with the first row of the partition.

1. In the query window, replace the script with the following to add aggregate functions:

```
-- ROWS UNBOUNDED PRECEDING
SELECT DISTINCT bc.Country, b.Title AS Book, bc.Downloads
    ,FIRST_VALUE(b.Title) OVER (PARTITION BY Country
        ORDER BY Downloads ASC ROWS UNBOUNDED PRECEDING) AS FewestDownloads
FROM dbo.BookConsumption AS bc
INNER JOIN dbo.Books AS b
    ON b.ID = bc.BookID
ORDER BY Country, Downloads
```

2. Select Run from the toolbar menu to execute the SQL command.





```sql
-- ROWS UNBOUNDED PRECEDING
SELECT DISTINCT bc.Country, b.Title AS Book, bc.Downloads
    ,FIRST_VALUE(b.Title) OVER (PARTITION BY Country
        ORDER BY Downloads ASC ROWS UNBOUNDED PRECEDING) AS FewestDownloads
FROM dbo.BookConsumption AS bc
INNER JOIN dbo.Books AS b
    ON b.ID = bc.BookID
ORDER BY Country, Downloads
```

| Country | Book | Downloads | FewestDownloads |
|---|---|---|---|
| Germany | Love Like That (The Romance Chr... | 1295.66666666667 | Harry Potter - The Ultimate Quiz Book |
| Germany | A Reaper at the Gates | 1418 | Harry Potter - The Ultimate Quiz Book |
| Germany | Harry Potter - The Ultimate Book... | 1565.66666666667 | Harry Potter - The Ultimate Quiz Book |
| Germany | Harry Potter - The Amazing Quiz ... | 1721.66666666667 | Harry Potter - The Ultimate Quiz Book |
| Germany | 101 Amazing Harry Potter Facts | 2410.33333333333 | Harry Potter - The Ultimate Quiz Book |
| Germany | Harry Potter - The Ultimate Quiz | 2812.66666666667 | Harry Potter - The Ultimate Quiz Book |
| Sweden | Burn for Me | 1.19999999999999 | Burn for Me |
| Sweden | Diamond Fire | 6 | Burn for Me |
| Sweden | Game Of Thrones The Quiz Book... | 8.19999999999999 | Burn for Me |
| Sweden | Harry Potter - The Complete Qui... | 11 | Burn for Me |
| Sweden | The World of Ice and Fire: The U... | 15.6 | Burn for Me |

In this query, we use the FIRST_VALUE analytic function to retrieve the book title with the fewest downloads, as indicated by the ROWS UNBOUNDED PRECEDING clause over the Country partition (1). The UNBOUNDED PRECEDING option set the window start to the first row of the partition, giving us the title of the book with the fewest downloads for the country within the partition.

In the result set, we can scroll through the list that of books by country, sorted by number of downloads in ascending order.