uday sai

4 Followers About

SQL for Data Science : Window Functions

uday sai Jul 1, 2020 · 6 min read

Window functions play an important role in the data extraction phase of Data Science projects and is an important skill to have in repertoire. The window function are executed after From, Where, Group By, Having clauses along with 'SELECT' clause. Unlike Group By clause window functions operates on a set of rows and generates result for each row. The set of rows is called as 'Window' on which the function operates.

A window function is differentiated in the query by using a over() clause, which differentiates window function from the other analytical and reporting function. A query can include multiple window functions with different window size. The over() have the following capabilities:

> *PARTITION BY*
>
> *ORDER BY*
>
> *ROWS/RANGE BETWEEN [START] AND [FINISH]*

Anatomy of window function

Let us explore the anatomy of the window function in detail:

PARTITION BY [*column_name*] : This clause is used to partition the table and the window function is applied with respect to each partition. For example, if partition is applied on a column having 10 unique values then window function is applied for each unique value.

ORDER BY [*column_name*] [*desc*|*asc*] : This clause is used to order the rows in each partition and the window function is applied in a row-wise manner.

RANGE | ROWS BETWEEN [PRECEDING | FOLLOWING] : This clause is used to specify the frame size in a window. This frame size is useful to modify the set of rows on which the window function operates. By default the window functions are applied on all the subsequent rows till current row. This frame may needed to be modified for some functions (as we see later). For now let us get familiarized with how to specify frame size.

ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW (This is default frame size which takes all the subsequent (syntactically represented as 'UNBOUNDED') rows till the CURRENT ROW (syntactical representation) in a particular partition)

ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING (now frame size is current row and 3 rows following the current row)

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING (rows from first row in the partition till the last row in the partition)

*Note: In all the above 4 examples **RANGE** can be used in the place of **ROWS**.*

The difference between ROWS and RANGE is the former is a physical operator which treats different rows having same column_name as different rows and the latter is a logical operator which treats all the rows having same column_name as a single entity. To illustrate this let us consider the following queries:

```
SELECT CustomerID, SalesOrderID, TotalDue, OrderDate,
        SUM(TotalDue) OVER(PARTITION BY CustomerID) CustomerTotal,
        SUM(TotalDue) OVER(PARTITION BY CustomerID ORDER BY OrderDate
            CustomerRunningTotal
FROM Sales.SalesOrderHeader
WHERE CustomerID = 11300;
```

	CustomerID	SalesOrderID	TotalDue	OrderDate	CustomerRunningTotal
6	11300	56487	38.664	2007-10-22 00:00:00.000	313.124
7	11300	58365	97.7483	2007-11-22 00:00:00.000	453.8347
8	11300	58370	42.9624	2007-11-22 00:00:00.000	453.8347
9	11300	60130	9.934	2007-12-16 00:00:00.000	463.7687
10	11300	61611	66.8194	2008-01-07 00:00:00.000	530.5881
11	11300	62876	38.664	2008-01-28 00:00:00.000	689.1004
12	11300	62896	119.8483	2008-01-28 00:00:00.000	689.1004
13	11300	63307	33.1279	2008-02-01 00:00:00.000	722.2283
14	11300	63770	2.5305	2008-02-08 00:00:00.000	756.2292
15	11300	63771	31.4704	2008-02-08 00:00:00.000	756.2292
16	11300	64195	23.7465	2008-02-15 00:00:00.000	779.9757
17	11300	64397	42.9624	2008-02-18 00:00:00.000	944.4439
18	11300	64417	121.5058	2008-02-18 00:00:00.000	944.4439
19	11300	65975	65.1729	2008-03-11 00:00:00.000	1009.6168

In the above picture we can see that RANGE is used as a default frame size and all the highlighted values in the picture represents that the 'sum' operator is applied on the columns having unique values and then applied in the order specified in 'ORDER BY' clause. In contrast in the below picture (in the second query) we can see that the 'SUM' operator is applied on each row despite these rows comprising of unique values for a column.

```
SELECT CustomerID, SalesOrderID, TotalDue, OrderDate,
        SUM(TotalDue) OVER(PARTITION BY CustomerID) CustomerTotal,
        SUM(TotalDue) OVER(PARTITION BY CustomerID ORDER BY SalesO
                  CustomerRunningTotal
FROM Sales.SalesOrderHeader
WHERE CustomerID = 11300;

SELECT CustomerID, SalesOrderID, TotalDue, OrderDate,
        SUM(TotalDue) OVER(PARTITION BY CustomerID) CustomerTotal,
        SUM(TotalDue) OVER(PARTITION BY CustomerID ORDER BY OrderD
                ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
              CustomerRunningTotal
FROM Sales.SalesOrderHeader
WHERE CustomerID = 11300;
```

	CustomerID	SalesOrderID	TotalDue	OrderDate	CustomerTotal	CustomerRunningTotal
6	11300	56487	38.664	2007-10-22 00:00:00.000	1658.0753	313.124
7	11300	58365	97.7483	2007-11-22 00:00:00.000	1658.0753	410.8723
8	11300	58370	42.9624	2007-11-22 00:00:00.000	1658.0753	453.8347
9	11300	60130	9.934	2007-12-16 00:00:00.000	1658.0753	463.7687
10	11300	61611	66.8194	2008-01-07 00:00:00.000	1658.0753	530.5881
11	11300	62876	38.664	2008-01-28 00:00:00.000	1658.0753	569.2521
12	11300	62896	119.8483	2008-01-28 00:00:00.000	1658.0753	689.1004
13	11300	63307	33.1279	2008-02-01 00:00:00.000	1658.0753	722.2283
14	11300	63770	2.5305	2008-02-08 00:00:00.000	1658.0753	724.7588
15	11300	63771	31.4704	2008-02-08 00:00:00.000	1658.0753	756.2292
16	11300	64195	23.7465	2008-02-15 00:00:00.000	1658.0753	779.9757

Image Source : ROW query

The window function are segregated as follows:

Value	Rank	Aggregations
LEAD()	ROW_NUMBER()	MIN()
LAG()	RANK()	MAX()
LAST_VALUE()	DENSE_RANK()	SUM()
FIRST_VALUE()	CUME_DIST()	AVG()
NTH_VALUE()	PERCENT_RANK()	COUNT()
	NTILE()	

Image Source : Window Functions

The value function is used to retrieve the values of the specified columns. Let us explore the above functions one by one:

LEAD(column_name,n): The LEAD() window function returns the value for the row after the current row in a partition. If no row exists, null is returned. The lead value can be applied on any column_name, 'n' represents a positive integer value and retrieves the value with 'n' specified rows.

LAG(column_name,n): The LAG() window function returns the value for the row before the current row in a partition. If no row exists, null is returned. The lag value can be applied on any column_name, 'n' represents a positive integer value and retrieves the value with 'n' specified rows.

FIRST_VALUE(): The FIRST_VALUE() window function returns the value of the specified expression with respect to the first row in the window frame. The frame size is specified using syntax like 'UNBOUNDED PRECEDING | UNBOUNDED FOLLOWING' is used to specify frame size as a sliding window.

LAST_VALUE() : The LAST_VALUE window function returns the value of the specified expression with respect to the last row in the window frame. he frame size is specified using syntax like 'UNBOUNDED PRECEDING | UNBOUNDED FOLLOWING' is used to

UNBOUNDED PRECEDING AND CURRENT ROW, we often need the frame size to be UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING.

ROW_NUMBER(): The ROW_NUMBER window function determines the ordinal number of the current row within its partition. The ORDER BY expression in the OVER clause determines the number. Each value is ordered within its partition. Rows with equal values for the ORDER BY expressions receive different row numbers nondeterministically.

RANK(): The RANK window function determines the rank of a value in a group of values. The ORDER BY expression in the OVER clause determines the value. Each value is ranked within its partition. Rows with equal values for the ranking criteria receive the same rank. Drill adds the number of tied rows to the tied rank to calculate the next rank and thus the ranks might not be consecutive numbers. For example, if two rows are ranked 1, the next rank is 3. The DENSE_RANK window function differs in that no gaps exist if two or more rows tie.

DENSE_RANK(): The DENSE_RANK () window function determines the rank of a value in a group of values based on the ORDER BY expression and the OVER clause. Each value is ranked within its partition. Rows with equal values receive the same rank. There are no gaps in the sequence of ranked values if two or more rows have the same rank.

CUME_DIST(): The CUME_DIST() window function calculates the relative rank of the current row within a window partition: (number of rows preceding or peer with current row) / (total rows in the window partition)

PERCENT_RANK(): The PERCENT_RANK () window function calculates the percent rank of the current row using the following formula: (x — 1) / (number of rows in window partition — 1) where x is the rank of the current row.

NTILE(): The NTILE window function divides the rows for each window partition, as equally as possible, into a specified number of ranked groups. The NTILE window function requires the ORDER BY clause in the OVER clause.

COUNT(): The COUNT() window function counts the number of input rows. COUNT(*) counts all of the rows in the target table if they do or do not include nulls. COUNT(expression) computes the number of rows with non-NULL values in a specific column or expression.

MAX(): The MAX() window function returns the maximum value of the expression across all input values. The MAX function works with numeric values and ignores NULL values.

MIN(): The MIN() window function returns the minimum value of the expression across all input values. The MIN function works with numeric values and ignores NULL values.

SUM(): The SUM() window function returns the sum of the expression across all input values. The SUM function works with numeric values and ignores NULL values.

Note: All most all the recent versions of databases including MySQL, PostgreSQL, Hive, Impala supports the window functions. Refer the below picture for in-depth details

Product	Support
Oracle*	with support since version 8i (1998)
PostgreSQL*	with support since version 8.4 (2009)
IBM Db2*	with support since version 9 for z/OS (2008)
Microsoft SQL Server*	with limited support since SQL Server 2005 extended support in SQL Server 2012
SAP/Sybase SQL Anywhere*	with support since version 12 (2010)
Firebird**	with limited support since version 3 (2016)
SQLite**	with support since version 3.25 (2018) extended support in version 3.28 (2019)
Amazon Redshift*	with support since version 1 released (2013)
MySQL*	with support since version 8.0.2 (2018)
MariaDB*	with support since version 10.2 (2017)

IBM Informix	with support since version 12.10 (2013)

Image Source : Databases Supported

References:

Aggregate Window Functions

Window functions operate on a set of rows and return a single value for each row from the underlying query. The OVER()...

drill.apache.org

What is the Difference between ROWS and RANGE?

UPDATE: For more from me about T-SQL Window Functions, check out my Pluralsight course! I have been speaking and...

auntkathisql.com

https://data-xtractor.com/blog/query-builder/window-functions-support/#:~:text=Oracle%20Database%20had%20window%20functions,MEDIAN%2C%20and%20FIRST%2FLAST.

Data Science Machine Learning Sql Data Extraction Data Analysis