|  |
| --- |
| EPAM Systems, RD Dep., RD Dep. |
| MTN.BI.03 SQL for Analysis  **Windowing Aggregate Functions** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| REVISION HISTORY | | | | | |
| Ver. | Description of Change | Author | Date | Approved | |
| Name | Effective Date |
| 1.0 | Initial Version | Aliaksei Belablotski | 15-Feb-2012 |  |  |
|  |  |  |  |  |  |

Contents

[1. Windowing Aggregate Functions 3](#_Toc317444335)

[1.1. Oracle Analytic Functions 3](#_Toc317444336)

[1.2. Essential Concepts of Analytic Functions 4](#_Toc317444337)

[1.2.1. Processing order 4](#_Toc317444338)

[1.2.2. Result set partitioning 4](#_Toc317444339)

[1.2.3. Window 4](#_Toc317444340)

[1.2.4. Current row 4](#_Toc317444341)

[1.3. Windowing Aggregation Functions 5](#_Toc317444342)

[1.3.1. Physical and Logical Offset 5](#_Toc317444343)

[1.3.2. Centered Aggregate Function 6](#_Toc317444344)

[1.3.3. Varying Window Size for Each Row 6](#_Toc317444345)

[1.4. LISTAGG Function 7](#_Toc317444346)

[1.5. NTH\_VALUE Function 7](#_Toc317444347)

# Windowing Aggregate Functions

## Oracle Analytic Functions

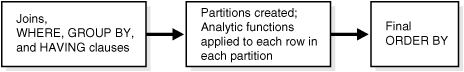
Oracle enhances SQL's analytical processing capabilities by introducing a new family of analytic SQL functions. These analytic functions enable you to calculate:

* Rankings and percentiles
* Moving window calculations
* Lag/lead analysis
* First/last analysis
* Linear regression statistics

To enhance performance, analytic functions can be parallelized: multiple processes can simultaneously execute all of these statements. These capabilities make calculations easier and more efficient, thereby enhancing database performance, scalability, and simplicity.

|  |  |
| --- | --- |
| **Type** | **Used For** |
| Ranking | Calculating ranks, percentiles, and n-tiles of the values in a result set. |
| **Windowing** | **Calculating cumulative and moving aggregates. Works with these functions: SUM, AVG, MIN, MAX, COUNT, VARIANCE, STDDEV, FIRST\_VALUE,LAST\_VALUE, and statistical functions. Note that the DISTINCT keyword is not supported in windowing functions except for MAX and MIN.** |
| Reporting | Works with: SUM, AVG, MIN, MAX, COUNT (with/without DISTINCT), VARIANCE, STDDEV, RATIO\_TO\_REPORT, and statistical functions. Note that the DISTINCT keyword may be used in those reporting functions that support DISTINCT in aggregate mode. |
| LAG/LEAD | Finding a value in a row a specified number of rows from a current row. |
| FIRST/LAST | First or last value in an ordered group. |
| Linear Regression | Calculating linear regression and other statistics. |
| Inverse Percentile | The value in a data set that corresponds to a specified percentile. |
| Hypothetical Rank and Distribution | The rank or percentile that a row would have if inserted into a specified data set. |

New elements build on existing SQL to allow flexible and powerful calculation expressions. With just a few exceptions, the analytic functions have these new elements. The processing flow is represented in figure below.



## Essential Concepts of Analytic Functions

### Processing order

Query processing using analytic functions takes place in three stages (see picture above). First, all joins, WHERE, GROUP BY and HAVING clauses are performed. **Second, the result set is made available to the analytic functions, and all their calculations take place.** Third, if the query has an ORDER BY clause at its end, the ORDER BY is processed to allow for precise output ordering.

### Result set partitioning

The analytic functions allow users to divide query result sets into groups of rows called partitions. Note that the term partitions used with analytic functions is unrelated to the table partitions feature. **Partitions are created after the groups defined with GROUP BY clauses, so they are available to any aggregate results such as sums and averages.** Partition divisions may be based upon any desired columns or expressions. A query result set may be partitioned into just one partition holding all the rows, a few large partitions, or many small partitions holding just a few rows each.

### Window

For each row in a partition, you can define a sliding window of data. This window determines the range of rows used to perform the calculations for the current row. Window sizes can be based on either a physical number of rows or a logical interval such as time. The window has a starting row and an ending row. Depending on its definition, the window may move at one or both ends. For instance, a window defined for a cumulative sum function would have its starting row fixed at the first row of its partition, and its ending row would slide from the starting point all the way to the last row of the partition. In contrast, a window defined for a moving average would have both its starting and end points slide so that they maintain a constant physical or logical range.

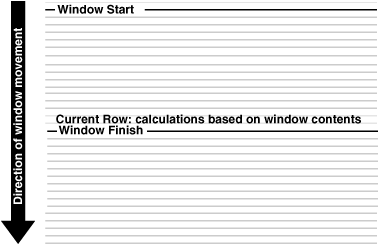
A window can be set as large as all the rows in a partition or just a sliding window of one row within a partition. **When a window is near a border, the function returns results for only the available rows, rather than warning you that the results are not what you want.**

**When using window functions, the current row is included during calculations, so you should only specify (n-1) when you are dealing with n items.**

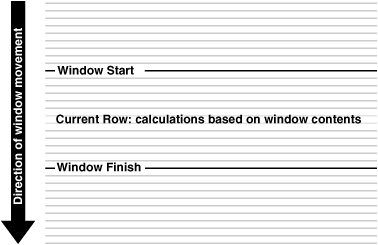
### Current row

**Each calculation performed with an analytic function is based on a current row within a partition.** The current row serves as the reference point determining the start and end of the window. For instance, a centered moving average calculation could be defined with a window that holds the current row, the six preceding rows, and the following six rows. This would create a sliding window of 13 rows.

For example, window for cumulative calculations will look like:



But window for calculating some averages will be slightly different:



## Windowing Aggregation Functions

Windowing functions can be used to compute cumulative and centered aggregates. They return a value for each row in the table, which depends on other rows in the corresponding window. With windowing aggregate functions, you can calculate moving and cumulative versions of SUM, AVERAGE, COUNT, MAX, MIN, and many more functions. They can be used only in the SELECT and ORDER BY clauses of the query. Windowing aggregate functions include the convenient FIRST\_VALUE, which returns the first value in the window; and LAST\_VALUE, which returns the last value in the window. **These functions provide access to more than one row of a table without a self-join.** The syntax of the windowing functions is:

analytic\_function([ arguments ])

OVER (analytic\_clause)

where analytic\_clause =

[ query\_partition\_clause ]

[ order\_by\_clause [ windowing\_clause ] ]

and query\_partition\_clause =

PARTITION BY

{ value\_expr[, value\_expr ]...

| ( value\_expr[, value\_expr ]... )

}

and windowing\_clause =

{ ROWS | RANGE }

{ BETWEEN

{ UNBOUNDED PRECEDING

| CURRENT ROW

| value\_expr { PRECEDING | FOLLOWING }

}

AND

{ UNBOUNDED FOLLOWING

| CURRENT ROW

| value\_expr { PRECEDING | FOLLOWING }

}

| { UNBOUNDED PRECEDING

| CURRENT ROW

| value\_expr PRECEDING

}

}

**Note that the DISTINCT keyword is not supported in windowing functions except for MAX and MIN.**

See also: [Analytic Functions Syntax Reference](http://docs.oracle.com/cd/B28359_01/server.111/b28286/functions001.htm#SQLRF20035)

### Physical and Logical Offset

**ROWS | RANGE** These keywords define for each row a window (a physical or logical set of rows) used for calculating the function result. The function is then applied to all the rows in the window. The window moves through the query result set or partition from top to bottom.

* ROWS specifies the window in physical units (rows).
* RANGE specifies the window as a logical offset.

If you specified **ROWS**:

* value\_expr is a physical offset. It must be a constant or expression and must evaluate to a positive numeric value.
* If value\_expr is part of the start point, then it must evaluate to a row before the end point.

If you specified **RANGE**:

* value\_expr is a logical offset. It must be a constant or expression that evaluates to a positive numeric value or an interval literal. Refer to "Literals" for information on interval literals.
* You can specify only one expression in the order\_by\_clause
* If value\_expr evaluates to a numeric value, then the ORDER BY expr must be a numeric or DATE datatype.
* If value\_expr evaluates to an interval value, then the ORDER BY expr must be a DATE datatype.

**If you omit the windowing\_clause entirely, then the default is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.**

A logical offset can be specified with constants such as RANGE 10 PRECEDING, or an expression that evaluates to a constant, or by an interval specification like RANGE INTERVAL N DAY/MONTH/YEAR PRECEDING or an expression that evaluates to an interval.

With logical offset, there can only be one expression in the ORDER BY expression list in the function, with type compatible to NUMERIC if offset is numeric, or DATE if an interval is specified.

An analytic function that uses the RANGE keyword can use multiple sort keys in its ORDER BY clause if it specifies either of these two windows:

* RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW. (The short form of this is RANGE UNBOUNDED PRECEDING, which can also be used.)
* RANGE BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING.

Window boundaries that do not meet these conditions can have only one sort key in the analytic function's ORDER BY clause.

### Centered Aggregate Function

Calculating windowing aggregate functions centered around the current row is straightforward. This example computes for all customers a centered moving average of sales for one week in late December 1999. It finds an average of the sales total for the one day preceding the current row and one day following the current row including the current row as well.

SELECT t.time\_id, TO\_CHAR (SUM(amount\_sold), '9,999,999,999')

AS SALES, TO\_CHAR(AVG(SUM(amount\_sold)) OVER

(ORDER BY t.time\_id

RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND

INTERVAL '1' DAY FOLLOWING), '9,999,999,999') AS CENTERED\_3\_DAY\_AVG

FROM sales s, times t

WHERE s.time\_id=t.time\_id AND t.calendar\_week\_number IN (51)

AND calendar\_year=1999

GROUP BY t.time\_id

ORDER BY t.time\_id;

**The starting and ending rows for each product's centered moving average calculation in the output data are based on just two days, since the window calculation cannot reach past the data retrieved by the query.** Users need to consider the different window sizes found at the borders of result sets: the query may need to be adjusted.

### Varying Window Size for Each Row

There are situations where it is useful to vary the size of a window for each row, based on a specified condition. For instance, you may want to make the window larger for certain dates and smaller for others.

SELECT T.TIME\_ID, T.DAY\_NAME, TO\_CHAR (SUM(AMOUNT\_SOLD), '9,999,999,999') AS SALES,

TO\_CHAR(AVG(SUM(AMOUNT\_SOLD)) OVER (ORDER BY T.TIME\_ID

RANGE BETWEEN INTERVAL '1' DAY PRECEDING AND

INTERVAL '1' DAY FOLLOWING), '9,999,999,999') AS CENTERED\_3\_DAY\_AVG,

TO\_CHAR(AVG(SUM(AMOUNT\_SOLD)) OVER (ORDER BY T.TIME\_ID

RANGE BETWEEN

(CASE WHEN T.DAY\_NAME = 'Monday' THEN (INTERVAL '2' DAY) ELSE (INTERVAL '1' DAY) END) PRECEDING

AND

INTERVAL '1' DAY FOLLOWING), '9,999,999,999') AS CENTERED\_3\_DAY\_AVG

FROM SALES S, TIMES T

WHERE S.TIME\_ID=T.TIME\_ID AND T.CALENDAR\_WEEK\_NUMBER IN (50, 51)

AND CALENDAR\_YEAR=1999

GROUP BY T.TIME\_ID, T.DAY\_NAME

ORDER BY T.TIME\_ID;

## LISTAGG Function

For a specified measure, LISTAGG orders data within each group specified in the ORDER BY clause and then concatenates the values of the measure column.

* As a single-set aggregate function, LISTAGG operates on all rows and returns a single output row.
* As a group-set aggregate, the function operates on and returns an output row for each group defined by the GROUP BY clause.
* As an analytic function, LISTAGG partitions the query result set into groups based on one or more expression in the query\_partition\_clause.

The arguments to the function are subject to the following rules:

* The measure\_expr can be any expression. Null values in the measure column are ignored.
* The delimiter\_expr designates the string that is to separate the measure values. This clause is optional and defaults to NULL.
* The order\_by\_clause determines the order in which the concatenated values are returned. The function is deterministic only if the ORDER BY column list achieved unique ordering.

The return data type is RAW if the measure column is RAW; otherwise the return value is VARCHAR2.

See also: [LISTAGG function in Oracle 11g release 2](http://www.oracle-developer.net/display.php?id=515)

## NTH\_VALUE Function

NTH\_VALUE returns the measure\_expr value of the nth row in the window defined by the analytic\_clause. The returned value has the data type of the measure\_expr.

{RESPECT | IGNORE} NULLS determines whether **null values of measure\_expr are included in or eliminated** from the calculation. The default is RESPECT NULLS.

Parameter n determines the nth row for which the measure value is to be returned. n can be a constant, bind variable, column, or an expression involving them, as long as it resolves to a positive integer. The function returns NULL if the data source window has fewer than n rows. If n is null, then the function returns an error.

FROM {FIRST | LAST} determines whether the **calculation begins at the first or last row of the window**. The default is FROM FIRST.

See also: [NTH\_VALUE syntax](http://docs.oracle.com/cd/E14072_01/server.112/e10592/functions112.htm)