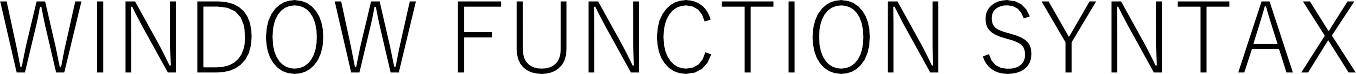


**Window functions** applies aggregate, ranking and analytic functions over a particular window (set of rows).

And **OVER** clause is used with window functions to define that window.

Give output one row per aggregation The rows maintain their separate identities

80



|  |  |  |
| --- | --- | --- |
| **fun**( ) |  | **OVER** |

**Select a function** Aggregate functions Ranking functions Analytic functions

SELECT column\_name(s),

**(** [ <PARTITION BY Clause> ] [ <ORDER BY Clause> ]

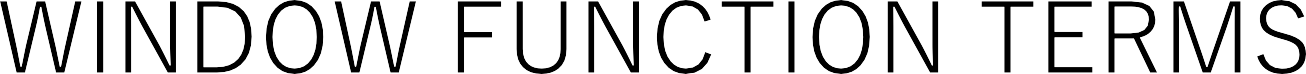
[ <ROW or RANGE Clause> ] **)**

FROM table\_name

**Define a Window** PARTITION BY ORDER BY ROWS



81







**Window function** applies aggregate, ranking and analytic functions over a particular window; for example, sum, avg, or row\_number

**Expression** is the name of the column that we want the window function operated on. This may not be necessary depending on what window function is used

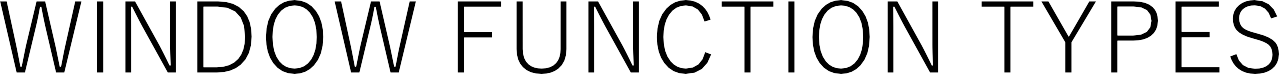
**OVER** is just to signify that this is a window function

**PARTITION BY** divides the rows into partitions so we can specify which rows to use to compute the window function

**ORDER BY** is used so that we can order the rows within each partition. This is optional and does not have to be specified

**ROWS** can be used if we want to further limit the rows within our partition. This is optional and usually not used

82



There is no official division of the SQL window functions into categories but high level we can divide into three types

**Value/Analytic**

LEAD LAG

FIRST\_VALUE

LAST\_VALUE

**Ranking**

ROW\_NUMBER RANK DENSE\_RANK PERCENT\_RANK

**Aggregate**

SUM AVG COUNT MIN MAX

**Window Functions**

83

**SELECT** new\_id, new\_cat,

**AGGREGATE FUNCTION**

**Example**

**SUM**(new\_id) **OVER**( PARTITION BY new\_cat ORDER BY new\_id ) AS "Total", **AVG**(new\_id) **OVER**( PARTITION BY new\_cat ORDER BY new\_id ) AS "Average", **COUNT**(new\_id) **OVER**( PARTITION BY new\_cat ORDER BY new\_id ) AS "Count", **MIN**(new\_id) **OVER**( PARTITION BY new\_cat ORDER BY new\_id ) AS "Min", **MAX**(new\_id) **OVER**( PARTITION BY new\_cat ORDER BY new\_id ) AS "Max" **FROM** test\_data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **new\_id** | **new\_cat** | **Total** | **Average** | **Count** | **Min** | **Max** |
| 100 | Agni | 300 | 150 | 2 | 100 | 200 |
| 200 | Agni | 300 | 150 | 2 | 100 | 200 |
| 500 | Dharti | 1200 | 600 | 2 | 500 | 700 |
| 700 | Dharti | 1200 | 600 | 2 | 500 | 700 |
| 200 | Vayu | 1000 | 333.33333 | 3 | 200 | 500 |
| 300 | Vayu | 1000 | 333.33333 | 3 | 200 | 500 |
| 500 | Vayu | 1000 | 333.33333 | 3 | 200 | 500 |

84

**SELECT** new\_id, new\_cat,

**SUM**(new\_id) **OVER**( ORDER BY new\_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Total", **AVG**(new\_id) **OVER**( ORDER BY new\_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Average", **COUNT**(new\_id) **OVER**( ORDER BY new\_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Count", **MIN**(new\_id) **OVER**( ORDER BY new\_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Min", **MAX**(new\_id) **OVER**( ORDER BY new\_id ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS "Max"

**FROM** test\_data

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **new\_id** | **new\_cat** | **Total** | **Average** | **Count** | **Min** | **Max** |
| 100 | Agni | 2500 | 357.14286 | 7 | 100 | 700 |
| 200 | Agni | 2500 | 357.14286 | 7 | 100 | 700 |
| 200 | Vayu | 2500 | 357.14286 | 7 | 100 | 700 |
| 300 | Vayu | 2500 | 357.14286 | 7 | 100 | 700 |
| 500 | Vayu | 2500 | 357.14286 | 7 | 100 | 700 |
| 500 | Dharti | 2500 | 357.14286 | 7 | 100 | 700 |
| 700 | Dharti | 2500 | 357.14286 | 7 | 100 | 700 |

**AGGREGATE FUNCTION**

**Example**

85



**NOTE**: Above we have used **ROWS**

which will give a SINGLE output based on all INPUT Vaused)

SELECT new\_id,

**RANKING FUNCTION**

**Example**

ROW\_NUMBER() OVER(ORDER BY new\_id) AS "ROW\_NUMBER", RANK() OVER(ORDER BY new\_id) AS "RANK",

DENSE\_RANK() OVER(ORDER BY new\_id) AS "DENSE\_RANK", PERCENT\_RANK() OVER(ORDER BY new\_id) AS "PERCENT\_RANK"

FROM test\_data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **new\_id** | **ROW\_NUMBER** | **RANK** | **DENSE\_RANK** | **PERCENT\_RANK** |
| 100 | 1 | 1 | 1 | 0 |
| 200 | 2 | 2 | 2 | 0.166 |
| 200 | 3 | 2 | 2 | 0.166 |
| 300 | 4 | 4 | 3 | 0.5 |
| 500 | 5 | 5 | 4 | 0.666 |
| 500 | 6 | 5 | 4 | 0.666 |
| 700 | 7 | 7 | 5 | 1 |

86

**SELECT** new\_id,

**ANALYTIC FUNCTION**

**Example**

**FIRST\_VALUE**(new\_id) **OVER**( ORDER BY new\_id) AS "FIRST\_VALUE", **LAST\_VALUE**(new\_id) **OVER**( ORDER BY new\_id) AS "LAST\_VALUE", **LEAD**(new\_id) **OVER**( ORDER BY new\_id) AS "LEAD",

**LAG**(new\_id) **OVER**( ORDER BY new\_id) AS "LAG"

**FROM** test\_data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **new\_id** | **FIRST\_VALUE** | **LAST\_VALUE** | **LEAD** | **LAG** |
| 100 | 100 | 100 | 200 | null |
| 200 | 100 | 200 | 200 | 100 |
| 200 | 100 | 200 | 300 | 200 |
| 300 | 100 | 300 | 500 | 200 |
| 500 | 100 | 500 | 500 | 300 |
| 500 | 100 | 500 | 700 | 500 |
| 700 | 100 | 700 | null | 500 |

87



**NOTE**: If you just want the single last value from whole column, use **ROWS** BETWEEN



**Offset the LEAD and LAG values by 2 in the output columns ?**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **INPUT** |  |  | **OUTPUT** |  |  |
|  |  |  |  |  |  |
| **new\_id** |  | **new\_id** | **LEAD** | **LAG** |  |
| 100 |  | 100 | 200 | NULL |  |
| 200 |  | 200 | 300 | NULL |  |
| 200 |  | 200 | 500 | 100 |  |
| 300 |  | 300 | 500 | 200 |  |
| 500 |  | 500 | 700 | 200 |  |
| 500 |  | 500 | NULL | 300 |  |
| 700 |  | 700 | NULL | 500 |  |
|  |  |  |  |  | 88 |

**SELECT** new\_id,

**ANALYTIC FUNCTION**

**Assignment**

**LEAD**(new\_id**, 2**) **OVER**( ORDER BY new\_id) AS "LEAD\_by2", **LAG**(new\_id**, 2**) **OVER**( ORDER BY new\_id) AS "LAG\_by2" **FROM** test\_data

|  |  |  |
| --- | --- | --- |
| **new\_id** | **LEAD\_by2** | **LAG\_by2** |
| 100 | 200 | null |
| 200 | 300 | null |
| 200 | 500 | 100 |
| 300 | 500 | 200 |
| 500 | 700 | 200 |
| 500 | null | 300 |
| 700 | null | 500 |