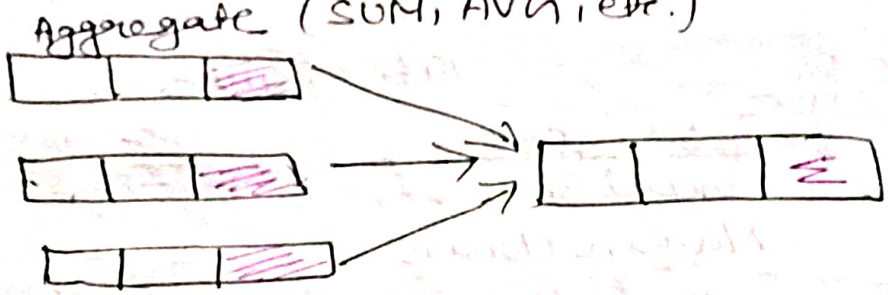
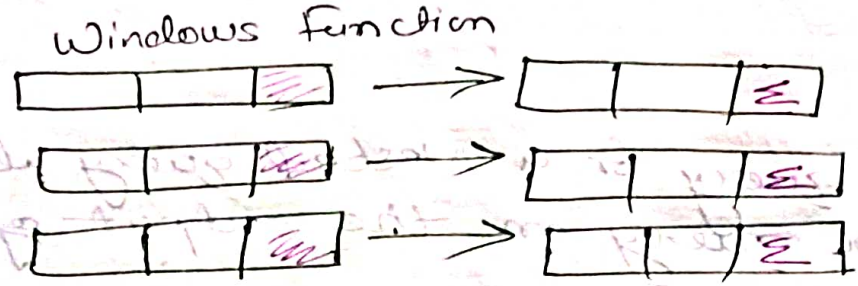
Windows function

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

And OVER clause is used with window function to define that window.





Window function Syntax

```
SELECT column_name(s),
    fun()   Over ( [<PARTION BY Clause>]
                   [<ORDER BY Clause>]
                   [<ROW or Range Clause>] )
FROM table_name
```

Select a function

- Aggregate functions
- Ranking functions
- Analytic functions.

Define a Window

- Partition By
- Order By
- Rows

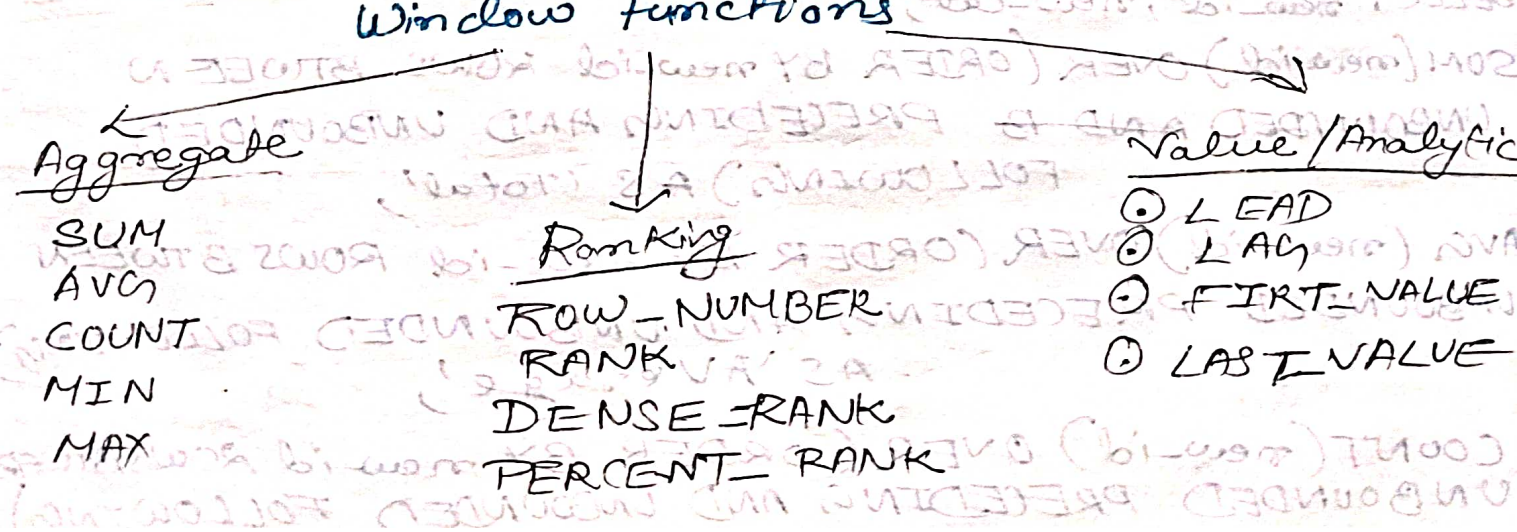
Windows function Terms

Let's look at some definitions:

- 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.

Windows Function Types

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



Aggregate Function Example

```
SELECT new_id, new_cat,
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
```

Output

new_id	new_cat	Total	Average	Count	Min	Max
100	Agni	100	100.00	1	100	100
200	Agni	300	150.00	2	100	200
500	Dharti	500	500	1	500	500
700	Dharti	1200	600	2	500	700
200	Vayu	900	200	1	200	200
300	Vayu	500	250	2	200	300
500	Vayu	1000	333	3	200	500

```
SELECT new_id, new_cat,
SUM (new_id) OVER (ORDER BY new_id ROWS BTWEEN
  UNBOUNDED ~~AND B~~ PRECEDING AND UNBOUNDED
                   FOLLOWING) AS 'Total',
AVG (new_id) OVER (ORDER BY new_id ROWS BTWEEN
UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
                   AS 'AVerage',
COUNT (new_id) OVER (ORDER BY new_id ROWSBTWEEN
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
		2500	357.1429	7	100	700
100	Agni	2500	〃	〃	〃	〃
200	Agni	2500	〃	〃	〃	〃
200	Vayu	2500	〃	〃	〃	〃
300	Vayu	2500	〃	〃	〃	〃
500	Dharti	2500	〃	〃	〃	〃
500	Vayu	2500	〃	〃	〃	〃
700	Dharti					

Note

Above we have used : 'ROWS BTWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING' which will give a SINGLE output based on all INPUT value / PARTITION (if used).

Ranking Function Example

```
SELECT new_id,
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.
```

Output

new_id	ROW_NUMBER	RANK	DENSE_RANK	PECENT_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

Analytic Function Example

```
SELECT new_id,
FIRST_VALUE (new_id) OVER (OVER ORDER BY ne
                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

Note

If you just want the single last value from whole column, use: 'ROWS BETWEEN UNBOUNDED PRECEDIN AND UNBOUNDED FOLLOWING'.

Quick Assignment: WINDOW FUNCTION

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

INPUT

new_id
100
200
200
300
500
500
700

OUTPUT

new_id	LEAD	LAG
100	200	null
200	300	null
200	500	100
300	500	200
500	200	300
500	null	300
700	null	500

```
SELECT new_id,
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.
```