What are window functions?

Window functions in SQL are a type of analytical function that perform calculations across a set of row that are related to the current row. called "window". A window function calculates a value for each row in the result set based on a subset of the row that are defined by a window specification.

The window specification is defined using the OVER() clause in SQL, which specifies the partioning and ordering of the row into groups based on a specific column or expression, while the ordering defines the order in which the rows are processed within each group.

```
SELECT branch, AVG(marks)
FROM marks
GROUP BY branch
```

Student_id	name	branch	marks
1	Nitish	EEE	82
2	Rishabh	EEE	91
3	Anukant	EEE	69
4	Rupesh	EEE	55
5	Shubham	CSE	78
6	Ved	CSE	43
7	Deepak	CSE	98
8	Arpan	CSE	95

1	Nitish	EEE	82
2	-	EEE	91
3	-	"	69
4	-	"	55

5	Shubham	CSE	78
6	-	"	43
7	-	"	98
8	-	"	95

branch	AVG(marks)
CSE	78, 5000
EEE	74, 2500

```
SELECT *
AVG(marks) OVER (PARTITION BY branch)
FROM marks
```

Std_id	name	branch	marks	Avg(Marks)
-	-	EEE	-	78.500
-	-	EEE	-	78.500
-	-	EEE	-	78.500
-	-	CSE	-	89.750
-	-	CSE	-	89.750
-	-	CSE	-	89.750
-	-	CSE	-	89.750

Aggregate Function with OVER()

Find all the students who have marks higher than the avg marks of their respectives branch.

```
SELECT * FROM(
SELECT *, Avg(marks) OVER (PARTITION BY branch) AS
'avgg' FROM campus.marks)t
WHERE t.marks > t.avgg
```

RANK/DENSE_RANK/ROW_Number

2) Find rank (RANK) of each branch according to the marks.

```
SELECT *,
RANK() OVER(PARTITION BY branch ORDER BY marks DESC)
FROM campus. marks
```

Std-id	name	branch	marks	rank()
7	-	CSE	98	1
8	-	CSE	95	2
5	-	CSE	78	3
6	-	CSE	43	4
9	-	ECE	95	1
12	-	ECE	95	1
10	-	ECE	88	3
11	-	ECE	81	4
2	-	MECH	91	1
1	-	MECH	82	2
3	-	MECH	69	3
4	-	MECH	55	4

If mark is same then both rank is same (like 1 and 1) and then next rank will be start from 3 not 2.

(Dense_rank)

mark	Rank	Dense_rank
95	1	1
95	1	1
89	3	2

difference

```
SELECT *,
~~RANK()~~ DENSE_RANK() OVER (PARTITION BY branch ORDER BY
                              marks DESC)
FROM campus. marks
```

(ROW_NUMBER)

```
SELECT *
ROW_NUMBER() OVER (PARTITION BY BRANCH)
FROM campus_marks
```

Std_id	Name	Branch	marks	Row_Numb
7	-	CSE	-	1
8	-	"	-	2
5	-	"	-	3
6	-	"	-	4
9	-	ECE	-	1
12	-	"	-	2
10	-	"	-	3
11	-	"	-	4
2	-	MECH	-	1
1	-	"	-	2
3	-	"	-	3
4	-	"	-	4

Row Number

Row Number

Row Number

FIRST_VALUE / LAST_VALUE / NTH_VALUE

First_value

first row show highest marks and create another column to show highest marks

```
SELECT *,
FIRST_VALUE (marks) over (ORDER BY marks DESC)
FROM campus_marks
```

Std_id	name	branch	marks	First value
7	-	-	98	98
6	-	-	92	98
3	-	-	91	98
2	-	-	90	98

Frames

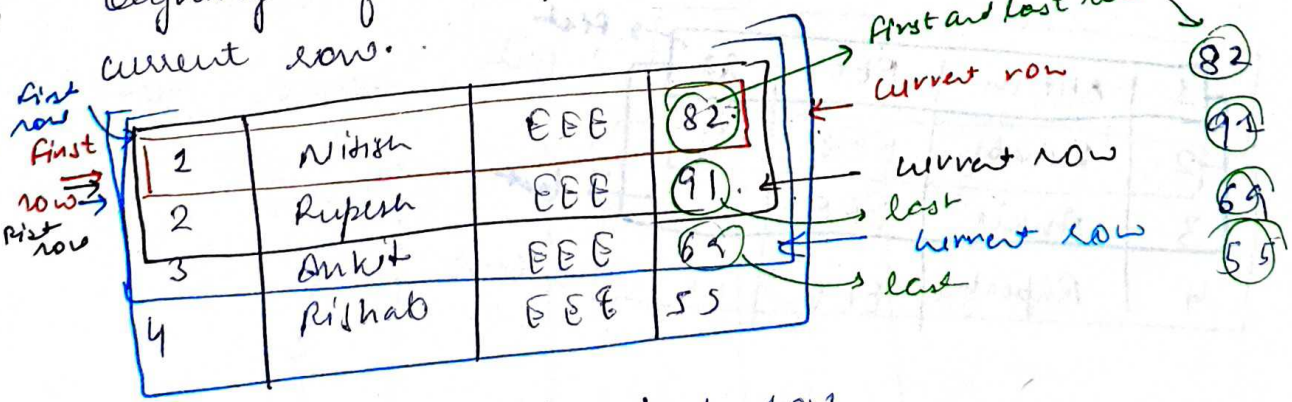
A frame in a window function is a subset of rows within the partition that determines the scope of the window function calculation. The frame is defined using a combination of two clauses in the window function ROWS and BETWEEN.

The ROWS clause specifies how many rows should be included in the frame relative to the current row. For example, ROWS PRECEDING means that the frame includes the current row and the three rows that precede it in the partition.

The BETWEEN clause specifies the boundaries of the frame.

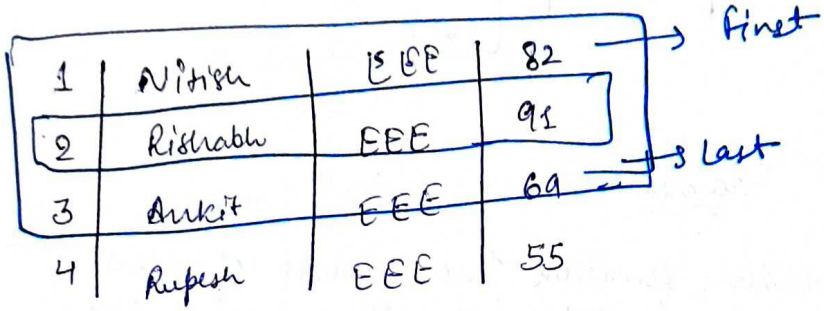
Example:

- ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW: (1st row) means that the frame includes all rows from the beginning of the partition up to and including the current row.

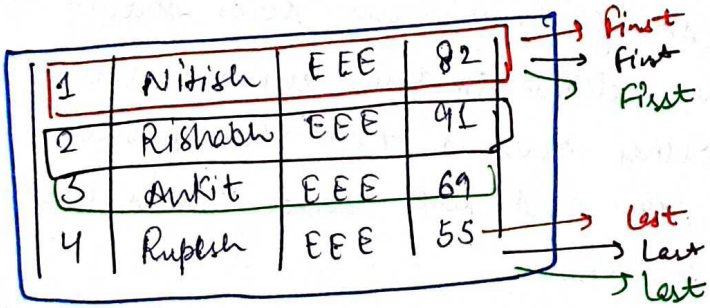


- we have to find Last row

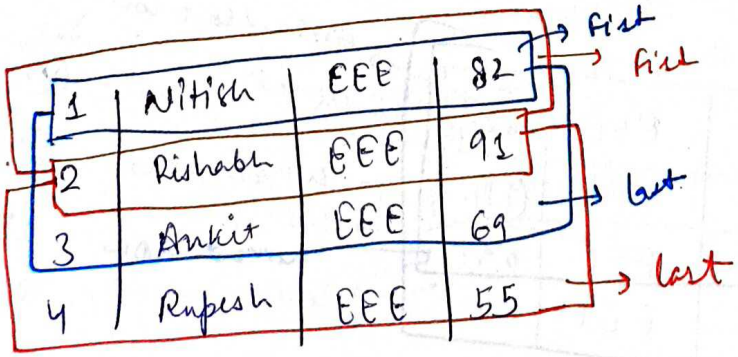
- ROWS between 1 PRECEDING AND 1 FOLLOWING: the frame includes the current row and the rows immediately before and after it.



- ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING: the frame include all rows in the partition.



- ROWS - BETWEEN 3 PRECEDING AND 2 FOLLOWING: the frame includes the current row and the three row before it and the two rows after it.



Last Value

```
SELECT *
LAST_VALUE (marks) OVER (PARTITION BY branch ORDER BY
                         marks DESC
ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING)
FROM marks.
```

Std_id	name	branch	marks	OVER(PART.. - -
7	-	CSE	95	73
8	-	CSE	95	73
5	-	CSE	73	73
6	-	EEE	72	54
12	-	EEE	63	54
.		EEE	54	54
.				
.				

NTH_VALUE

```
SELECT * ,
NTH_VALUE (name, 2) OVER (PARTITION BY branch ORDER BY
                          marks DESC
ROWS BETWEEN UNBOUNDED PROCEDING
AN UNBOUNDED FOLLOWING)
FROM marks.
```

Std_id	name	branch	marks	2nd value
7	-	CSE	91	95
8	-	CSE	95	95
5	-	CSE	73	95
6	-	EEE	63	54
12	-	EEE	54	54
.		EEE	29	54
.				
.				

1. Find name and branch of Topper.

```
SELECT name, branch, marks
FROM (
SELECT *,
FIRST_VALUE (name) OVER (PARTITION BY branch ORDER BY
                         marks DESC) AS 'Topper'
FROM campus. marks) t
WHERE name = t. Topper
```

name	branch	marks
Deepak	CSE	98
Vinay	ECE	95
Rishabh	EEE	91
Prashant	MECH	75

OR

```
SELECT name, branch, mark FROM (SELECT *,
LAST_VALUE (name) OVER W AS 'Topper'
FROM campus. marks )t
WHERE name = t. Topper

WINDOW W AS (PARTITION BY branch ORDER BY marks
             DESC ROWS BETWEEN UNBOUNDED PRECEDING
             AND UNBOUNDED FOLLOWING)
```

LEAD & LAG

LAG

```
SELECT *,
LAG (marks) OVER (ORDER BY student_id)
FROM marks
```

Student-id	Name	branch	marks	LAG(marks)
1	Nitish	EEE	82	NULL
2	Rishabh	EEE	91	82
3	Ankit	EEE	69	91
4	Rupesh	CSE	55	69
5	Shubham	CSE	75	55
6	Ved	CSE	78	75

LEAD

```
SELECT *,
LEAD (marks) OVER (ORDER BY student_id)
FROM marks
```

Student-id	Name	branch	marks	LEAD (marks)
1	Nitish	EEE	82	91
2	Rishabh	EEE	91	69
3	Ankit	EEE	69	55
4	Rupesh	CSE	55	75
5	Shubham	CSE	75	78
6	Ved	CSE	78	NULL