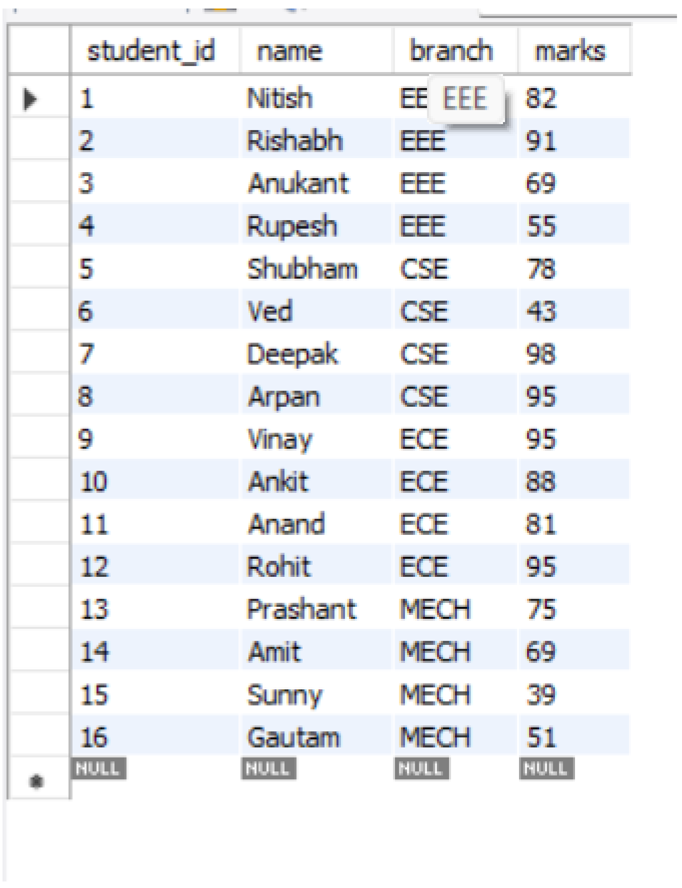
What are Window Functions?

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

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

Data:



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
9	Vinay	ECE	95
10	Ankit	ECE	88
11	Anand	ECE	81
12	Rohit	ECE	95
13	Prashant	MECH	75
14	Amit	MECH	69
15	Sunny	MECH	39
16	Gautam	MECH	51
NULL	NULL	NULL	NULL

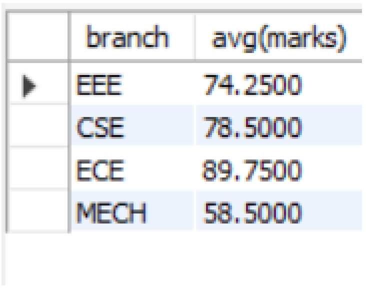
DIFFERENCE BETWEEN GROUPBY AND WINDOW_FUNCTION

GROUP BY:

```
Select branch,avg(marks)

From database.table

Groupby branch
```



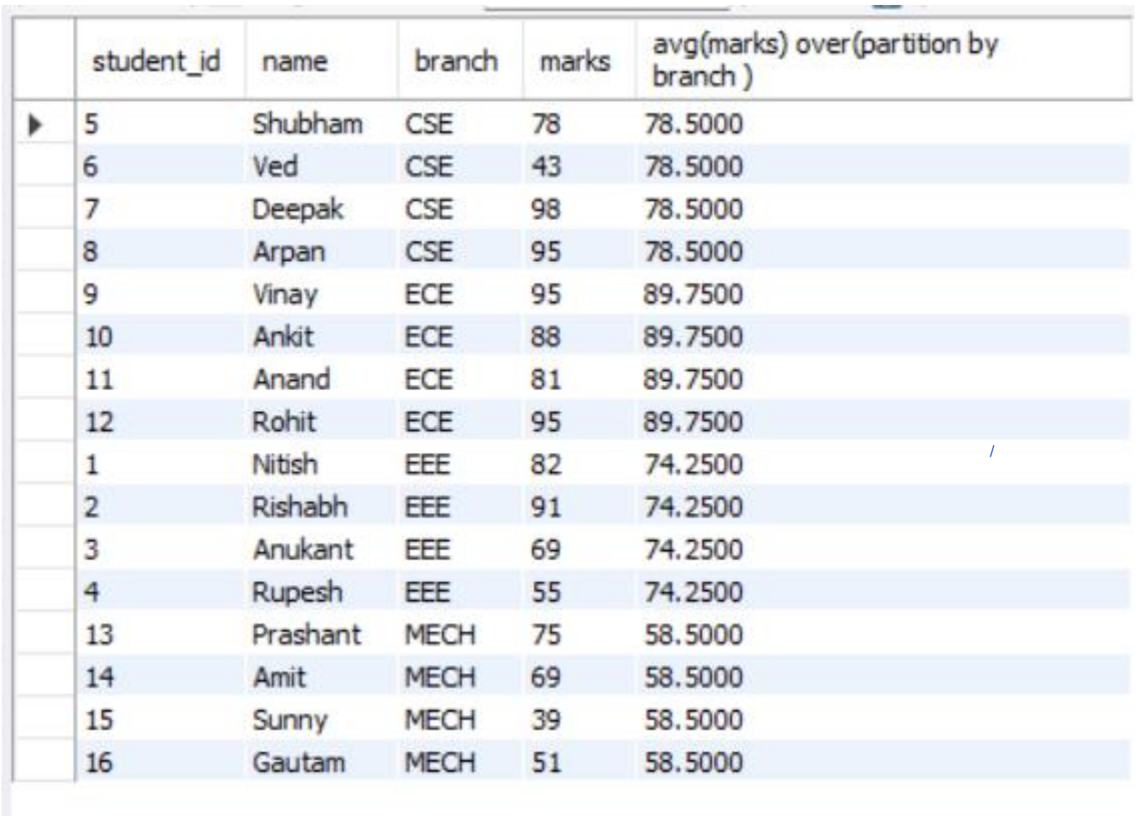
branch	avg(marks)
EEE	74.2500
CSE	78.5000
ECE	89.7500
MECH	58.5000

WINDOW FUNCTION:

```
Select *,

avg(marks) over(partition by branch )

from marks
```

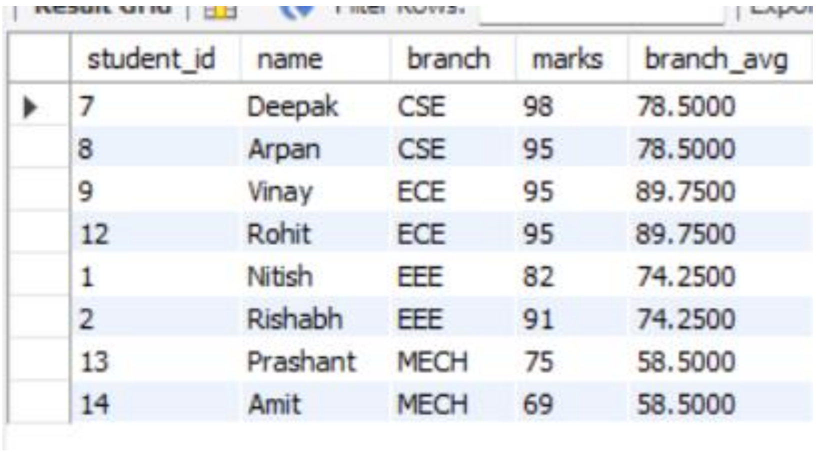


student_id	name	branch	marks	avg(marks) over(partition by branch)
5	Shubham	CSE	78	78.5000
6	Ved	CSE	43	78.5000
7	Deepak	CSE	98	78.5000
8	Arpan	CSE	95	78.5000
9	Vinay	ECE	95	89.7500
10	Ankit	ECE	88	89.7500
11	Anand	ECE	81	89.7500
12	Rohit	ECE	95	89.7500
1	Nitish	EEE	82	74.2500
2	Rishabh	EEE	91	74.2500
3	Anukant	EEE	69	74.2500
4	Rupesh	EEE	55	74.2500
13	Prashant	MECH	75	58.5000
14	Amit	MECH	69	58.5000
15	Sunny	MECH	39	58.5000
16	Gautam	MECH	51	58.5000

Aggregate Function with OVER()

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

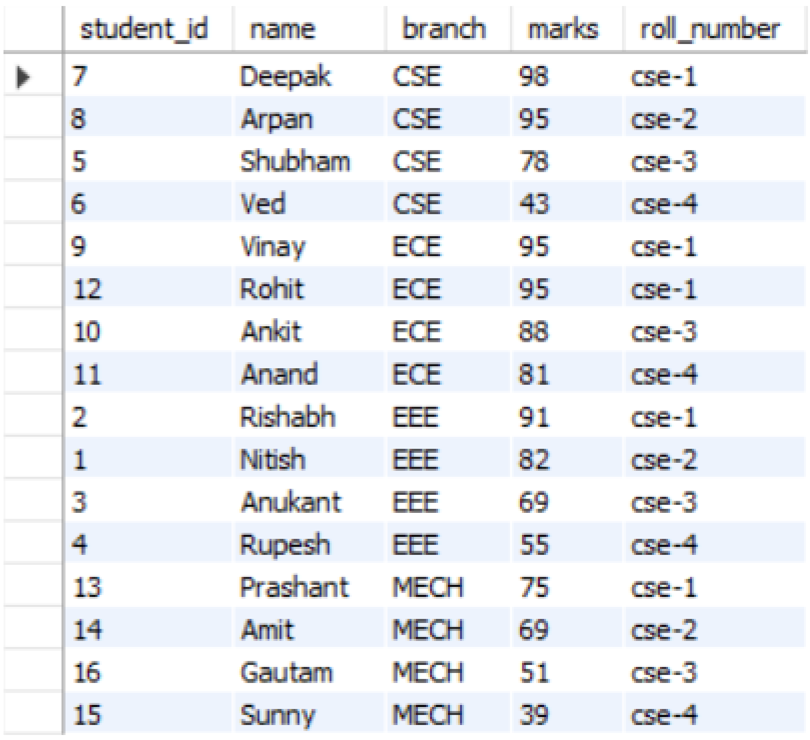
```
select * from (select *,
                        avg(marks) over(partition by branch) as 'branch_avg'
                        from marks) t
where marks>t.branch_avg
```



student_id	name	branch	marks	branch_avg
7	Deepak	CSE	98	78.5000
8	Arpan	CSE	95	78.5000
9	Vinay	ECE	95	89.7500
12	Rohit	ECE	95	89.7500
1	Nitish	EEE	82	74.2500
2	Rishabh	EEE	91	74.2500
13	Prashant	MECH	75	58.5000
14	Amit	MECH	69	58.5000

RANK/DENSE_RANK/ROW_NUMBER

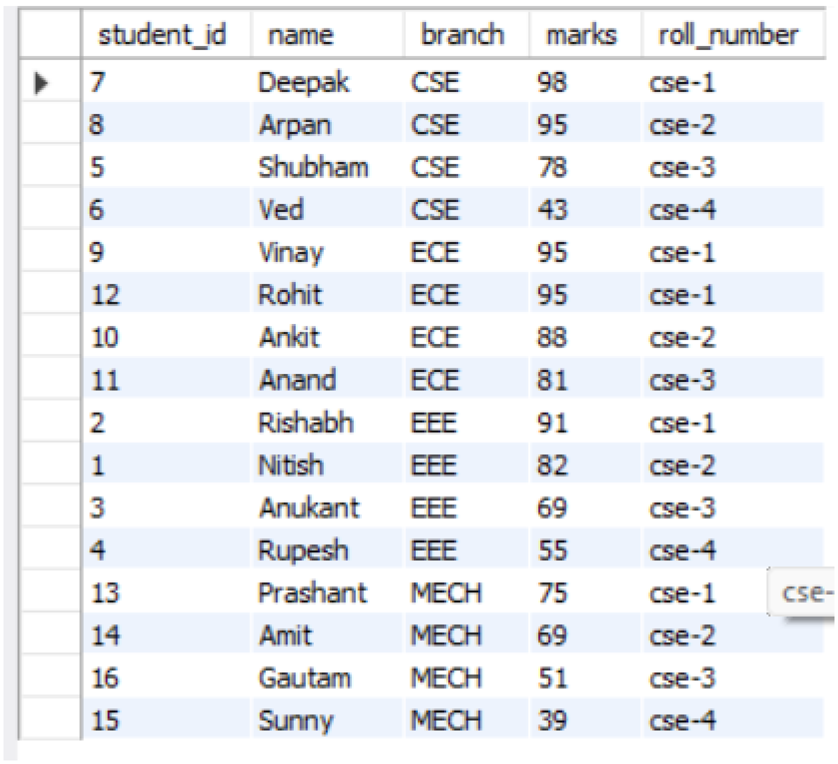
Create roll no from branch and marks



student_id	name	branch	marks	roll_number
7	Deepak	CSE	98	cse-1
8	Arpan	CSE	95	cse-2
5	Shubham	CSE	78	cse-3
6	Ved	CSE	43	cse-4
9	Vinay	ECE	95	cse-1
12	Rohit	ECE	95	cse-1
10	Ankit	ECE	88	cse-3
11	Anand	ECE	81	cse-4
2	Rishabh	EEE	91	cse-1
1	Nitish	EEE	82	cse-2
3	Anukant	EEE	69	cse-3
4	Rupesh	EEE	55	cse-4
13	Prashant	MECH	75	cse-1
14	Amit	MECH	69	cse-2
16	Gautam	MECH	51	cse-3
15	Sunny	MECH	39	cse-4

RANK:

```
select *,
concat('cse', '-' ,rank()
over(partition by branch
order by marks desc)) as
'roll_number'
from marks
```



student_id	name	branch	marks	roll_number
7	Deepak	CSE	98	cse-1
8	Arpan	CSE	95	cse-2
5	Shubham	CSE	78	cse-3
6	Ved	CSE	43	cse-4
9	Vinay	ECE	95	cse-1
12	Rohit	ECE	95	cse-1
10	Ankit	ECE	88	cse-2
11	Anand	ECE	81	cse-3
2	Rishabh	EEE	91	cse-1
1	Nitish	EEE	82	cse-2
3	Anukant	EEE	69	cse-3
4	Rupesh	EEE	55	cse-4
13	Prashant	MECH	75	cse-1
14	Amit	MECH	69	cse-2
16	Gautam	MECH	51	cse-3
15	Sunny	MECH	39	cse-4

DENSE_RANK:

select *,

concat('cse', '-' ,dense_rank() over(partition by branch order by marks desc)) as 'roll_number'

from marks

NOTE: DIFFERENCE IN RANK AND DENSE_RANK CLEARLY SEE.

FIRST_VALUE/LAST VALUE/NTH_VALUE

FIRST_VALUE:

Find the branch toppers

Query:

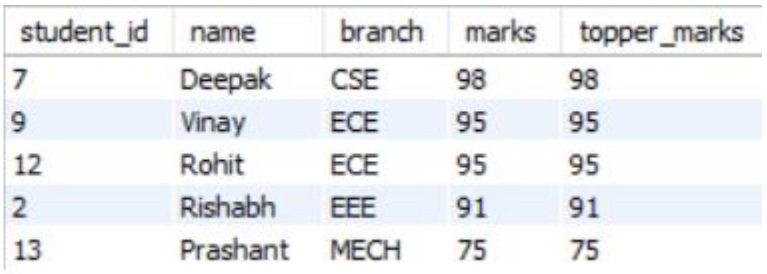
select * from (select *,

first_value(marks) over(partition by branch order by marks desc) as 'topper_marks'

from marks) t

where marks=t.topper_marks

RESULT:



student_id	name	branch	marks	topper_marks
7	Deepak	CSE	98	98
9	Vinay	ECE	95	95
12	Rohit	ECE	95	95
2	Rishabh	EEE	91	91
13	Prashant	MECH	75	75

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 3 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.

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

• ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING: the frame includes the current row and the row immediately before and after it.

• ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING: the frame includes all rows in the partition.

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

LAST_VALUE:

Find the last guy of each branch .

QUERY:

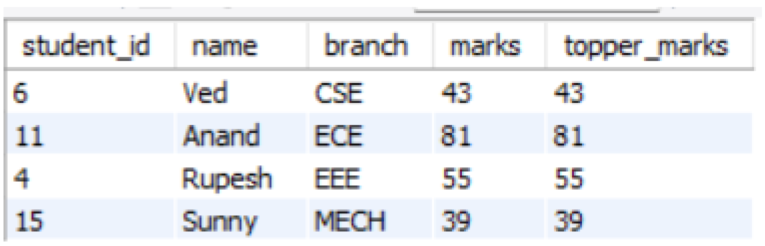
```
select * from (select *,

last_value(marks) over(partition by branch order by marks desc

rows between unbounded preceding and unbounded following) as 'topper_marks'

from marks) t

where marks=t.topper_marks
```



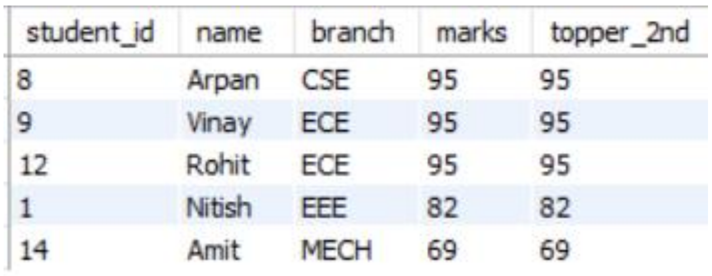
student_id	name	branch	marks	topper_marks
6	Ved	CSE	43	43
11	Anand	ECE	81	81
4	Rupesh	EEE	55	55
15	Sunny	MECH	39	39

NTH_VALUE:

Find the 2nd last guy of each branch.

QUERY:

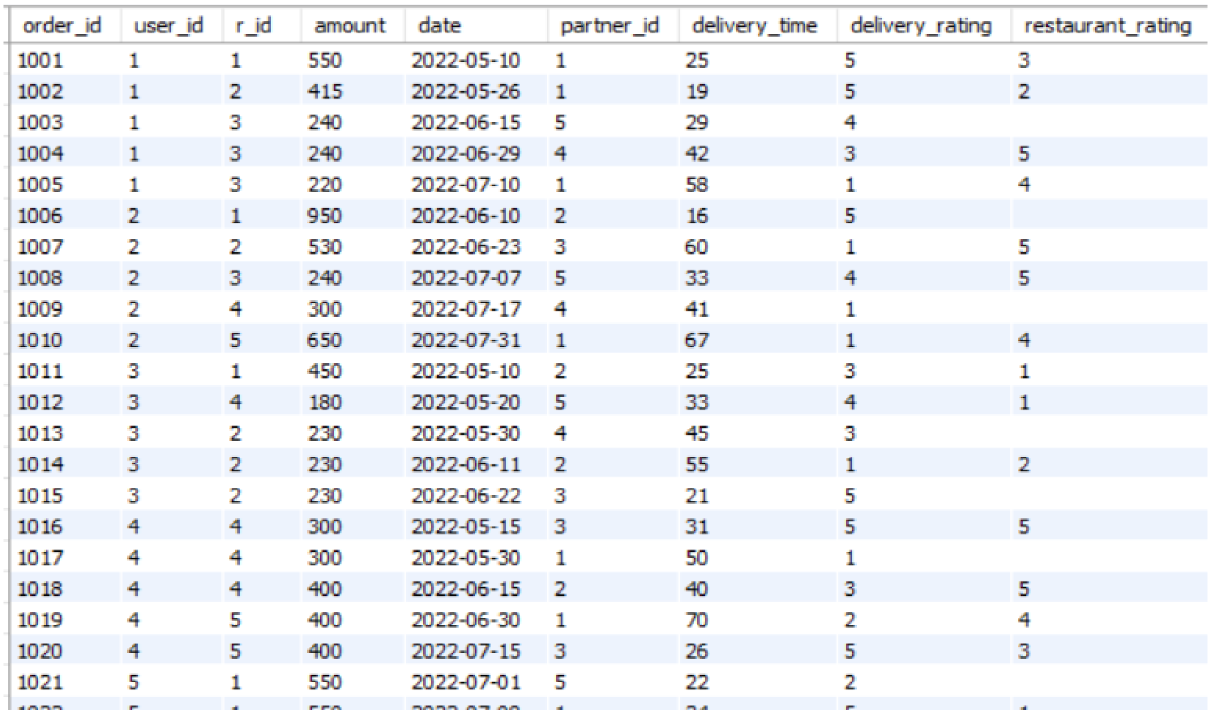
```
select * from(select *,
        nth_value(marks,2) over(partition by branch order by marks desc
        rows between unbounded preceding and unbounded following) as 'topper_2nd'
        From marks) t
where marks = t.topper_2nd
```



student_id	name	branch	marks	topper_2nd
8	Arpan	CSE	95	95
9	Vinay	ECE	95	95
12	Rohit	ECE	95	95
1	Nitish	EEE	82	82
14	Amit	MECH	69	69

LEAD & LAG:

DATA:

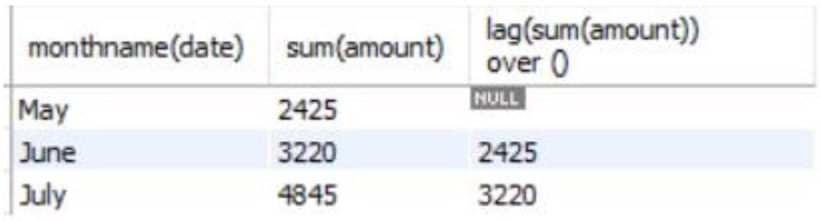


order_id	user_id	r_id	amount	date	partner_id	delivery_time	delivery_rating	restaurant_rating
1001	1	1	550	2022-05-10	1	25	5	3
1002	1	2	415	2022-05-26	1	19	5	2
1003	1	3	240	2022-06-15	5	29	4	
1004	1	3	240	2022-06-29	4	42	3	5
1005	1	3	220	2022-07-10	1	58	1	4
1006	2	1	950	2022-06-10	2	16	5	
1007	2	2	530	2022-06-23	3	60	1	5
1008	2	3	240	2022-07-07	5	33	4	5
1009	2	4	300	2022-07-17	4	41	1	
1010	2	5	650	2022-07-31	1	67	1	4
1011	3	1	450	2022-05-10	2	25	3	1
1012	3	4	180	2022-05-20	5	33	4	1
1013	3	2	230	2022-05-30	4	45	3	
1014	3	2	230	2022-06-11	2	55	1	2
1015	3	2	230	2022-06-22	3	21	5	
1016	4	4	300	2022-05-15	3	31	5	5
1017	4	4	300	2022-05-30	1	50	1	
1018	4	4	400	2022-06-15	2	40	3	5
1019	4	5	400	2022-06-30	1	70	2	4
1020	4	5	400	2022-07-15	3	26	5	3
1021	5	1	550	2022-07-01	5	22	2	

LAG:

```
select monthname(date),
sum(amount),
lag(sum(amount)) over ()
from `orders (1)`
group by monthname(date)
```

RESULT:

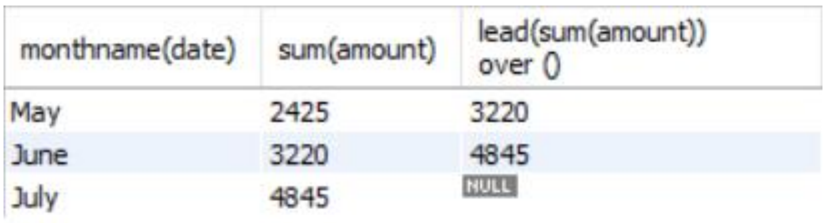


monthname(date)	sum(amount)	lag(sum(amount)) over ()
May	2425	NULL
June	3220	2425
July	4845	3220

Lead:

```
select monthname(date),
sum(amount),
lag(sum(amount)) over ()
from `orders (1)`
group by monthname(date)
```

Result:

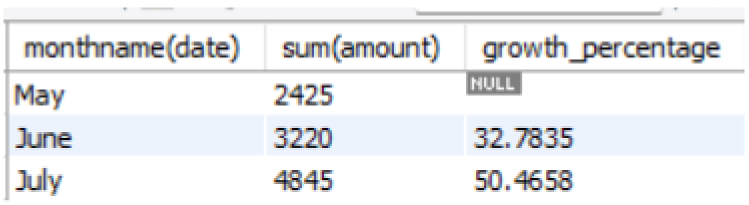


monthname(date)	sum(amount)	lead(sum(amount)) over ()
May	2425	3220
June	3220	4845
July	4845	NULL

Find the MoM revenue growth of Zomato.

QUERY:

```
select monthname(date),
sum(amount),
((sum(amount) - lag(sum(amount)) over())/ lag(sum(amount)) over())*100 as 'growth_percentage'
from `orders (1)`
group by monthname(date)
```



monthname(date)	sum(amount)	growth_percentage
May	2425	NULL
June	3220	32.7835
July	4845	50.4658

HAPPY LEARNING

Yogeshchouhan263@gmail.com