What are Window Function ?

Thursday, October 5, 2023 1:38 PM

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.

Group By
Select branch, Avg(marks) from marks
Group by branch;

Window fun
Select *, avg(marks)
Over(partition by branch)
from marks;

GROUP BY returns group where Window Function returns row by rows

student_id	name	branch	marks	avg_m
5	Shubham	CSE	78	78.5000
6	Ved	CSE	43	78.5000
7	Deepak	CSE	98	78.5000
8	Arpan	CSE	95	78.5000
12	Rohit	ECE	95	89.7500
9	Vinay	ECE	95	89.7500
10	Ankit	ECE	88	89.7500
11	Anand	ECE	81	89.7500
2	Rishabh	EEE	91	74.2500
3	Anukant	EEE	69	74.2500
4	Rupesh	EEE	55	74.2500
1	Nitish	EEE	82	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

return rows

AGGREGATE FUNCTION WITH OVER()

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

```
SELECT * FROM (SELECT *,AVG(marks) OVER(PARTITION BY branch)
avg_branch FROM marks) t
WHERE t.marks > t.avg_branch
```

RANK/DENSE_RANK/ROW_Number

It doesn't need orderby because of db engine.

```
SELECT *,
CONCAT(branch,'-',RANK() OVER(PARTITION BY branch ORDER BY marks DESC)),
DENSE_RANK() OVER(PARTITION BY branch ORDER BY marks DESC),
ROW_NUMBER() OVER(PARTITION BY branch ORDER BY marks DESC)
FROM marks
```

Find top 2 most paying customers of each month

```
SELECT * FROM (SELECT MONTHNAME(date) month, user_id,SUM(amount) AS total,
                RANK() OVER(PARTITION BY MONTHNAME(date) ORDER BY SUM(amount) DESC) month_rank
    FROM orders
    GROUP BY MONTHNAME(date),user_id
    ORDER BY MONTH(date)) t
    WHERE t.month_rank < 3
    ORDER BY month DESC,month_rank ASC
```

FIRST_VALUE/LAST_VALUE/NTH_VALUE

```
SELECT * ,
FIRST_VALUE(marks) OVER(ORDER BY marks DESC),
LAST_VALUE(marks) OVER(ORDER BY marks DESC
                ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING),
NTH_VALUE(name,2) OVER(ORDER BY marks DESC
                ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM marks
```

They are frames

2nd highest

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.

HIGHEST MARKS WITH NAME WITH ALTERNATIVE WAY

```
SELECT *
FROM (
   SELECT *,
      FIRST_VALUE(name) OVER w AS topper_name,
      FIRST_VALUE(marks) OVER w AS topper_marks
   FROM marks
   WINDOW w AS (PARTITION BY branch ORDER BY marks DESC)
) t
WHERE t.name = t.topper_name AND t.marks = t.topper_marks;
```

LEAD & LAG

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

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

EXAMPLE

Find the MoM revenue growth of Zomato

```
SELECT MONTHNAME(date),SUM(amount),
((SUM(amount)-LAG(SUM(amount)) OVER (ORDER BY MONTH(date)))/LAG(SUM(amount)) OVER (ORDER BY MONTH(date)))*100
FROM orders
GROUP BY MONTHNAME(date)
ORDER BY MONTH(date) ASC
```

MONTHNAME(date)	SUM(amount)	((SUM(amount)-LAG(SUM(amount)) OVER (ORDER BY MONTH(date)))/LAG(SUM(amount)) OVER (ORDER BY MONTH(date)))*100
May	2425	NULL
June	3220	32.7835
July	4845	50.4658

RANKING

Q. TOP 5 BatsMan from each batting ipl team

```
SELECT * FROM campusx.ipl;

SELECT * FROM (SELECT BattingTeam,batter,SUM(batsman_run) AS 'total_runs',
DENSE_RANK() OVER(PARTITION BY BattingTeam ORDER BY SUM(batsman_run)
DESC) AS 'rank_within_team'
FROM ipl
GROUP BY BattingTeam,batter) t
WHERE t.rank_within_team < 6
ORDER BY t.BattingTeam,t.rank_within_team;
```

Cumulative sum

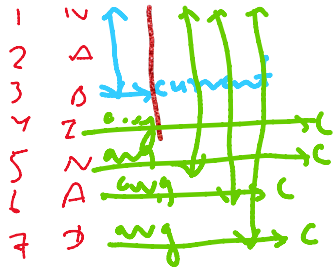
Cumulative sum is another type of calculation that can be performed using window functions. A cumulative sum calculates the sum of a set of values up to a given point in time, and includes all previous values in the calculation.

Cumulative average

Cumulative average is another type of average that can be calculated using window functions. A cumulative average calculates the average of a set of values up to a given point in time, and includes all previous values in the calculation.

Running Average

A running average in a window function is a way to calculate the average of a set of values, but it's done continuously as you move through a sequence of data points. Imagine you have a list of numbers, and you want to know the average of the numbers within a certain group or window.



CUMMULATIVE SUM and CUMMULATIVE AVERAGE and RUNNING AVERAGE

Q. Virat Kholi run,avg_run,running_avg

```
SELECT * FROM (SELECT
CONCAT("Match-",CAST(ROW_NUMBER() OVER(ORDER BY ID) AS CHAR)) AS 'match_no',
SUM(batsman_run) AS 'runs_scored',
SUM(SUM(batsman_run)) OVER w AS 'career_runs',
AVG(SUM(batsman_run)) OVER w AS 'career_avg',
AVG(SUM(batsman_run)) OVER(ROWS BETWEEN 9 PRECEDING AND CURRENT ROW) AS 'rolling_avg'
FROM ipl
WHERE batter = 'V Kohli'
GROUP BY ID
WINDOW w AS (ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)) t
```

Percent of total

Percent of total refers to the percentage or proportion of a specific value in relation to the total value. It is a commonly used metric to represent the relative importance or contribution of a particular value within a larger group or population.

Q.What is the Percent of Total sell of res_id 1

```
SELECT f_name,
(total_value/SUM(total_value) OVER())*100 AS 'percent_of_total'
FROM (SELECT f_id,SUM(amount) AS 'total_value' FROM orders t1
JOIN order_details t2
ON t1.order_id = t2.order_id
WHERE r_id = 1
GROUP BY f_id) t
JOIN food t3
ON t.f_id = t3.f_id
ORDER BY (total_value/SUM(total_value) OVER())*100 DESC
```

Percent change

Percent change is a way of expressing the difference between two values as a percentage of the original value. It is often used to measure how much a value has increased or decreased over a given period of time, or to compare two different values.

*Percent Change = ((new_value-old_value)/old_value)*100*

```
SELECT YEAR(Date),QUARTER(Date),SUM(views) AS 'views',
((SUM(views) - LAG(SUM(views)) OVER(ORDER BY
YEAR(Date),QUARTER(Date)))/LAG(SUM(views)) OVER(ORDER BY
YEAR(Date),QUARTER(Date)))*100 AS 'Percent_change'
FROM youtube_views
GROUP BY YEAR(Date),QUARTER(Date)
ORDER BY YEAR(Date),QUARTER(Date);
```

PERCENTILES AND QUANTILES

A Quantile is a measure of the distribution of a dataset that divides the data into any number of equally sized intervals. For example, a dataset could be divided into deciles (ten equal parts), quartiles (four equal parts), percentiles (100 equal parts), or any other number of intervals.

Each quantile represents a value below which a certain percentage of the data falls. For example, the 25th percentile (also known as the first quartile, or Q1) represents the value below which 25% of the data falls. The 50th percentile (also known as the median) represents the value below which 50% of the data falls, and so on.

```
--Find the median marks of all the students
SELECT *,
PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY marks) OVER() AS 'median_marks',
--Find branch wise median of student marks.
PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY marks) OVER(PARTITION BY branch) AS 'median_marks
--Percentile_Continious
PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY marks) OVER(PARTITION BY branch) AS 'median_marks_cont'
FROM marks;
```

PERCENTILE_CONT calculates the continuous percentile value, which returns the interpolated value between adjacent data points. In other words, it estimates the percentile value by assuming that the values between data points are distributed uniformly. This function returns a value that may not be present in the original dataset.

PERCENTILE_DISC, on the other hand, calculates the discrete percentile value, which returns the value of the nearest data point. This function returns a value that is present in the original dataset.

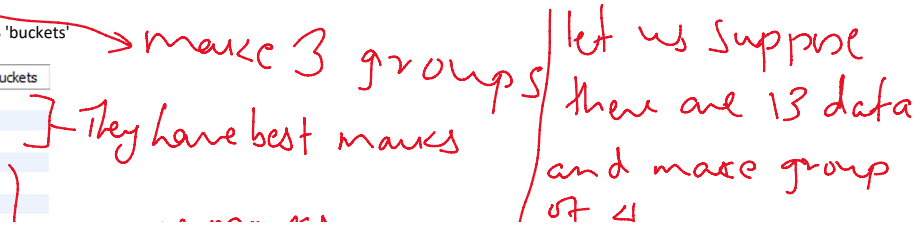
For example if we have 1,2,3,4,4,5

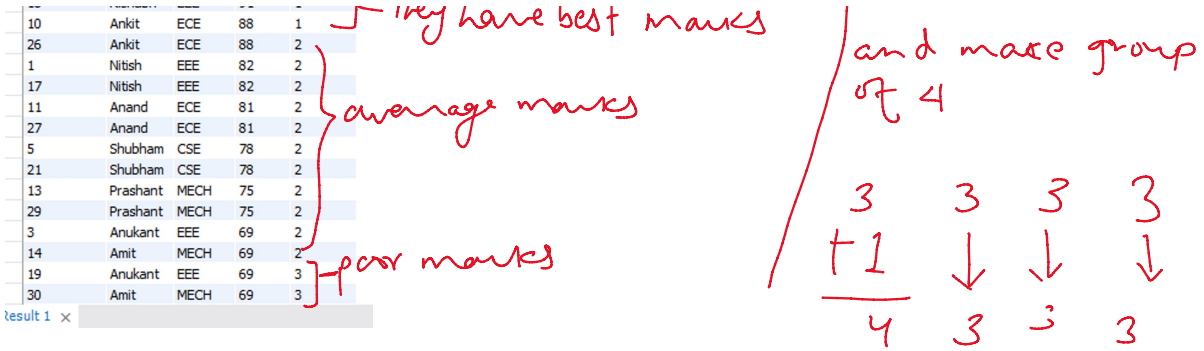
Segmentation

Segmentation using NTILE is a technique in SQL for dividing a dataset into equal-sized groups based on some criteria or conditions, and then performing calculations or analysis on each group separately using window functions.

```
SELECT *,
NTILE(3) OVER(ORDER BY marks DESC) AS 'buckets'
FROM marks;
```

student_id	name	branch	marks	buckets
2	Rishabh	EEE	91	1
18	Rishabh	EEE	91	1
10	Ankit	ECE	88	1
26	Ankit	ECE	88	2
1	Nitish	EEE	82	2
17	Nitish	EEE	82	2





10	Ankit	ECE	88	1
26	Ankit	ECE	88	2
1	Nitish	EEE	82	2
17	Nitish	EEE	82	2
11	Anand	ECE	81	2
27	Anand	ECE	81	2
5	Shubham	CSE	78	2
21	Shubham	CSE	78	2
13	Prashant	MECH	75	2
29	Prashant	MECH	75	2
3	Anukant	EEE	69	2
14	Amit	MECH	69	2
19	Anukant	EEE	69	3
30	Amit	MECH	69	3

Result 1

```
SELECT brand_name,model,price,
CASE
      WHEN bucket = 1 THEN 'budget'
   WHEN bucket = 2 THEN 'mid-range'
   WHEN bucket = 3 THEN 'premium'
END AS 'phone_type'
FROM (SELECT brand_name,model,price,
NTILE(3) OVER(PARTITION BY brand_name ORDER BY price) AS 'bucket'
FROM smartphones) t;
```

google	Google Pixel 3a XL	15999	budget
google	Google Pixel 4	20120	budget
google	Google Pixel 5A	32999	mid-range
google	Google Pixel 5	36000	mid-range
google	Google Pixel 6	40480	mid-range
google	Google Pixel 7 5G	53100	premium
google	Google Pixel 6 Pro	54300	premium
honor	Honor Play 30	9999	budget
honor	Honor X6	13999	budget

cumulative distribution

The cumulative distribution function is used to describe the probability distribution of random variables. It can be used to describe the probability for a discrete, continuous or mixed variable. It is obtained by summing up the probability density function and getting the cumulative probability for a random variable

"What percentage of the rows in the data set have a value less than or equal to the current row?"

Percentage=(Total Number of Rows in DatasetNumber of Rows with Value≤Current Row's Value)/(Total Number of Rows in Dataset)×100%

```
SELECT * FROM (SELECT *,
CUME_DIST() OVER(ORDER BY marks) AS 'Percentile_Score'
FROM marks) t
WHERE t.Percentile_Score > 0.90;
```

Partition By multiple columns

```
SELECT * FROM (SELECT source,destination,airline,AVG(price) AS 'avg_fare',
DENSE_RANK() OVER(PARTITION BY source,destination ORDER BY AVG(price)) AS 'rank'
FROM flights
GROUP BY source,destination,airline) t
WHERE t.rank < 2
```

How to connect SQL database with Python

```
Import mysql.connector
mydb = mysql.connector.connect(
host = "localhost"
user = "root"
password="1234"
)
--creating an instance of 'cursor' class which is used to execute the 'SQL' statements in 'Python'
cursor = mydb.cursor()

--Show databse
cursor.execute("SHOW DATABASE")

for x in cursor:
print(x)
```