Window Functions: A comprehensive guide to mastery.

Introduction:

In the analytic world, problems can easily be solved using basic Structured Query Language(SQL). But often times, some analytical questions require advanced SQL. One of the advanced concepts of SQL is the window functions, which help to create and author SQL queries that will work on rows of the table (windows) and also to perform aggregations and ranking of rows.

This article provides an easy and comprehensive guide to window functions in SQL, syntax, and case studies of analytical questions that can be solved using SQL window functions.

Table of Contents:

Window Functions: A comprehensive guide to mastery.

Prerequisites:

In order to fully grasp the concept of windows functions explained in this article in general, you need to have a background knowledge in SQL, its basic concepts, and how to write basic SQL queries, and sub-queries.

Definition of window functions

Window functions are classified as part of the advanced concept and topic in SQL, and often times a lot of data analysts and newbies that are just learning how to write SQL queries usually see window functions as a difficult concept. The overall goal of this technical article is to prove that notion is wrong.

What is a window in SQL?

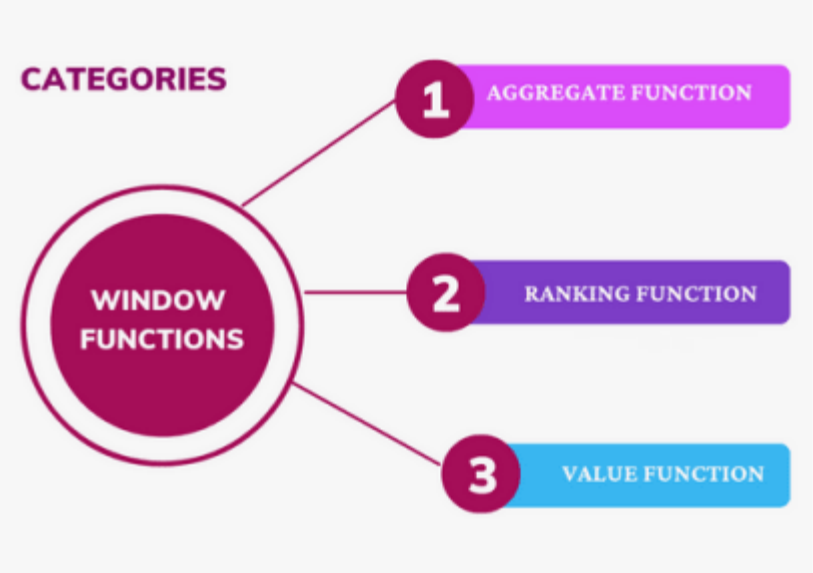
In SQL, a window refers to a group of rows. These group of rows are somehow related.

Categories of Window Functions

Windows functions are simply the functions that help you to aggregate, rank and compare group of rows in SQL without retuning a single output. What this definition simply means is that unlike
regular aggregate functions that will perform calculations on the rows of the table and thereafter return a single value, window functions on the other hand will perform the same calculation but
instead of grouping or aggregating the rows, the rows still maintain their position.

Windows functions are generally classified into three (3) categories namely:
i. Aggregate window functions
ii. Ranking window functions
iii. Value window functions

Window Functions: A comprehensive guide to mastery.



Syntax of window functions

The syntax of a basic SQL query using window functions is always in the form.

```
-- Window Function

SELECT [selected columns],

window_functions (column)

OVER ([PARTITION BY column] [ORDER BY column]) AS alias_name

table;
```

From the above syntax, the first word is **SELECT**, which helps us to query the table, and after that the required columns are specified. The next is **window_fucntion** which can either be an aggregate, ranking or value function, then the column that the window function will be effected on is specified inside the bracket. The **OVER** keyword specify the window (the set of rows that the window functions will be applied on). Then the **PARTITION BY** and **ORDER BY** clauses are specified, after which the table that the operation will be carried out on is specified.

Window Functions: A comprehensive guide to mastery.

PARTITION BY AND ORDER BY

PARTITION BY and ORDER BY are two function clauses that you will be using regularly when writing SQL queries that involves windows function. The function of the clauses is specified below.

- **PARTITION BY:** group the rows in the table into groups based on the specified column.
- **ORDER BY:** organize the rows in the table either in ascending or descending order.

Aggregate Window Functions

Brief Overview
The aggregate window functions basically perform aggregation on the window, but unlike regular aggregate functions, the singular value that is returned is based on the group of rows (windows).

Syntax
The syntax of aggregate window functions is basically similar to the syntax that I specified above for window functions. The only difference here is that in place of window functions the
aggregate functions are used.

```
-- Aggregate Window Function

SELECT [selected columns],

aggregate_functions (column)

OVER ([PARTITION BY column] [ORDER BY column]) AS alias_name

table;
```

SUM(), AVG(), MIN(), MAX(), COUNT()
The window aggregate functions are used to find the total or sum, average, minimum, maximum, count of the values in the window.

Window Functions: A comprehensive guide to mastery.

The aggregate window functions are used when finding the running total, running average and so on and so forth.

Running total or average in this concept simply refers to the calculation involving a particular row and every other rows above it. In order to achieve this, both ** PARTITION BY** and **ORDER BY** are used. But in order to calculate sum, average and so on for just the window or the group of rows, you need to use only **PARTITION BY** clause.

Ranking Window Functions

Brief Overview
The ranking window functions compares the value of current row with other rows in the window, and then return the rank or position of the current row with respect to the other values in the window.

Syntax
The syntax of ranking window functions is similar to the syntax that I specified for window functions. The only difference here is that in place of window functions the ranking functions are used and also the ranking window functions can be used without specifying the column in the bracket.

That is:

rank_fucntion() and not ~~rank_function(column)~~

```
-- Ranking Window Function

SELECT [selected columns],

ranking_functions ()

OVER ([PARTITION BY column] [ORDER BY column]) AS alias_name

table;
```

RANK(), DENSE_RANK(), ROW_NUNMBER(), NTILE()

The major ranking functions are **rank()**, **dense_rank()**, and **row_number()**. It is important to note that the three major ranking function do the same task but in

Window Functions: A comprehensive guide to mastery.

different ways and also, when you are using ranking functions, it is very important to include **ORDER BY** in the query, without which your query would not run.

i. Rank(): rank the rows in the window based on a given order and in a case where a given row has the same value as another row or rows, the identical rows are given the same rank, and then the next rank will be skipped.

ii. Dense_rank: rank the rows in the window based on a given order, and in case where a given row has the same value as another row or rows, the identical rows are given the same rank, but the next rank will not be skipped.

iii. Row_number: assign unique row number to the items in the window. That is two rows cannot have the same row number.

An additional ranking function is the **NTILE()** which determines the quartile that a row falls into, the **NTILE()** require the a parameter which specifies the type of quartile to be used.
Parameter is the value(s) that is specified in the bracket.
That is in the example NTILE(4), 4 here indicates the parameter.

Value Window Functions

Brief Overview
The value functions compare the value of a current row with rows before or after it. The two major function here are the **LAG()** and **LEAD() **functions.

Syntax
LAG(), LEAD(), FIRST_VALUE(), LAST_VALUE(), NTH_VALUE()
i. **LAG():** Determines the item before a particular row and assign the value of the preceding item to the current row.

ii. **LEAD():** Determines the item after a particular row and assign the value of the item to the current row.

LAG() and LEAD() functions require 3 parameters namely:

i. **Column:** column that the value will be gotten from.
ii. **Position:** The number of rows where the value will be gotten from.
iii. **Offset:** The offset is the value that will be return in case the current row is the first of the last in the window. In a case where the current value is the last or first in the window, the query will return NULL. But in order to prevent that, offset that can be added.

Window Functions: A comprehensive guide to mastery.

The other functions are pretty straight forward. The **first_value()** determine the first value in the window and it requires one parameter which is the column to be used as the determining factor.

Case studies of questions answered using window functions.

In order to follow through with the case study, you need to follow these steps:
i. Open the database management tool of your choice. (Like Microsoft SQL Server Management Studio)

ii. Create a table called student with the query:

```
create table student
(
id int primary key,
dept varchar(50),
score int
);
```

iii. Populate the table you just created.

```
INSERT INTO student (id, dept, score)
VALUES
(1,'Computer Science', 89),
(2,'Finance', 45),
(3,'Engineering', 90),
(4,'Engineering', 89),
(5,'Finance', 75),
(6,'Computer Science', 66),
(7,'Engineering', 89),
(8,'Computer Science', 81),
(9,'Computer Science', 52),
(10,'Engineering', 73),
(11,'Finance', 82),
(12,'Computer Science', 75);
```

iv. Check the status of the table:
SELECT * FROM student;

Window Functions: A comprehensive guide to mastery.


```
SELECT * FROM student;
```

id	dept	score
1	Computer Science	89
2	Finance	45
3	Engineering	90
4	Engineering	89
5	Finance	75
6	Computer Science	66
7	Engineering	89
8	Computer Science	81
9	Computer Science	52
10	Engineering	73
11	Finance	82
12	Computer Science	75

Aggregate windows functions

Running Score (Total) for each department.

Window Functions: A comprehensive guide to mastery.

```sql
-- Running Total

SELECT id, dept, score,
SUM(score) OVER (PARTITION BY dept ORDER BY score)
AS 'Running Total'
FROM student;
```

id	dept	score	Running Total
9	Computer Science	52	52
6	Computer Science	66	118
12	Computer Science	75	193
8	Computer Science	81	274
1	Computer Science	89	363
10	Engineering	73	73
4	Engineering	89	251
7	Engineering	89	251
3	Engineering	90	341
2	Finance	45	45
5	Finance	75	120
11	Finance	82	202

Total Score for each department

```sql
-- Running Total

SELECT id, dept, score,
SUM(score) OVER (PARTITION BY dept)
AS 'Running Total'
FROM student;
```

Window Functions: A comprehensive guide to mastery.

id	dept	score	Running Total
1	Computer Science	89	363
8	Computer Science	81	363
9	Computer Science	52	363
6	Computer Science	66	363
12	Computer Science	75	363
7	Engineering	89	341
10	Engineering	73	341
3	Engineering	90	341
4	Engineering	89	341
5	Finance	75	202
2	Finance	45	202
11	Finance	82	202

Other aggregation functions like MIN(), MAX(), and AVG() can also be used in place of SUM() based on the question to be solved.

```sql
-- Aggregate window functions

SELECT id, dept, score,
SUM(score) OVER (PARTITION BY dept) AS 'Running Total',
AVG(score) OVER (PARTITION BY dept) AS 'Departmental Average',
MIN(score) OVER (PARTITION BY dept) AS 'Departmental Minimum',
MAX(score) OVER (PARTITION BY dept) AS 'Departmental Maximum',
COUNT(score) OVER (PARTITION BY dept) AS 'Student in Department'
FROM student;
```

id	dept	score	Running Total	Departmental Average	Departmental Minimum	Departmental Maximum	Student in Department
1	Computer Science	89	363	72	52	89	5
8	Computer Science	81	363	72	52	89	5
9	Computer Science	52	363	72	52	89	5
6	Computer Science	66	363	72	52	89	5
12	Computer Science	75	363	72	52	89	5
7	Engineering	89	341	85	73	90	4
10	Engineering	73	341	85	73	90	4
3	Engineering	90	341	85	73	90	4
4	Engineering	89	341	85	73	90	4
5	Finance	75	202	67	45	82	3
2	Finance	45	202	67	45	82	3
11	Finance	82	202	67	45	82	3

Window Functions: A comprehensive guide to mastery.

Ranking windows functions

```sql
-- Ranking window functions

SELECT id, dept, score,
RANK() OVER (PARTITION BY dept ORDER BY score) AS 'Rank',
DENSE_RANK() OVER (PARTITION BY dept ORDER BY score) AS 'Dense Rank',
ROW_NUMBER() OVER (PARTITION BY dept ORDER BY score) AS 'Row Number'
FROM student;
```

id	dept	score	Rank	Dense Rank	Row Number
9	Computer Science	52	1	1	1
6	Computer Science	66	2	2	2
12	Computer Science	75	3	3	3
8	Computer Science	81	4	4	4
1	Computer Science	89	5	5	5
10	Engineering	73	1	1	1
4	Engineering	89	2	2	2
7	Engineering	89	2	2	3
3	Engineering	90	4	3	4
2	Finance	45	1	1	1
5	Finance	75	2	2	2
11	Finance	82	3	3	3

Value windows functions

```sql
-- Value window functions

SELECT id, dept, score,
LAG(score,1,0) OVER (PARTITION BY dept ORDER BY score DESC) AS 'Lag',
LEAD(score,1,0) OVER (PARTITION BY dept ORDER BY score DESC) AS 'Lead',
FIRST_VALUE(score) OVER (PARTITION BY dept ORDER BY dept) AS 'First Score',
LAST_VALUE(score) OVER (PARTITION BY dept ORDER BY dept) AS 'Last Score'
FROM student;
```

Window Functions: A comprehensive guide to mastery.

id	dept	score	Lag	Lead	First Score	Last Score
1	Computer Science	89	0	81	89	52
8	Computer Science	81	89	75	89	52
12	Computer Science	75	81	66	89	52
6	Computer Science	66	75	52	89	52
9	Computer Science	52	66	0	89	52
3	Engineering	90	0	89	90	73
4	Engineering	89	90	89	90	73
7	Engineering	89	89	73	90	73
10	Engineering	73	89	0	90	73
11	Finance	82	0	75	82	45
5	Finance	75	82	45	82	45
2	Finance	45	75	0	82	45

Conclusion

In this article, I have tried in as much as possible to provide a foundation for you as far as window functions is concerned, and with continuous practice you will soon become very good at writing
SQL queries using the concepts that I have discussed in this article. And if at all you run into error while writing your queries you can always refer back to the article for clarifications.