2021-07-16T18:00:00+02:00
Ignacio L. Bisso

Top 10 SQL Window Functions Interview Questions

Many interesting job positions require SQL skills – and that includes window functions, which are not commonly taught in online courses. In this article, I will cover the top window function questions for every experience level.

If you're going for a job interview for an advanced SQL position or for intermediate to advanced data analyst positions, you'll probably be asked about your knowledge of SQL window functions. Don't panic! Although these functions aren't commonly covered in online courses, we've got the answers right here.

Common Job Interview Questions About SQL Window Functions

The idea of this article is to help you prepare for questions on different window functions subtopics. We cannot give you the exact question you will receive, but we can be fairly certain about the topics to which the questions will point.

Don't just read about window functions – practice what you're learning. I recommend LearnSQL.com's Window Functions course. It's a great hands-on way to dig into using analytical functions to power up your SQL.

In some cases, the question can be very open, leaving the decision about what window function subtopic to cover entirely to you. In this

case, you should know the relative importance of each subtopic. For starters, you should be prepared for an open question like:

1. What Is a Window Function in SQL?

Window functions are SQL functions that operate on a set of records called the "window" or the "window frame". The "window" is a set of rows that are somehow related to the row currently being processed by the query (e.g. all rows before the current row, 5 rows before the current row, or 3 rows after the current row).

Window functions are similar to aggregate functions in that they compute statistics for a group of rows. However, window functions do not collapse rows; they keep the details of individual rows.

Window functions can be organized into the following four categories: aggregate functions, ranking functions, analytic functions, and distribution functions.

Aggregate functions are those that you use with `GROUP BY`. This includes:

- `COUNT()` counts the number of rows within the window.
- `AVG()` calculates the average value for a given column for all the records in the window.
- `MAX()` obtains the maximum value of a column for all the records in the window.
- `SUM()` returns the sum of all values in a given column within the window.

In the ranking category:

- `ROW_NUMBER()` returns the position of the row in the result set.
- `RANK()` ranks rows based on a given value. When two rows are in the same position, it awards them the same rank and leaves the next position empty (e.g. 1, 2, 3, 3, 5...).
- `DENSE_RANK()` also ranks rows by a given value, but it does not leave the next position empty (e.g. 1, 2, 3, 3, 4, 5...).

For detailed information, see this article about ranking functions.

In the analytic category, the functions `LEAD()`, `LAG()` or `FIRST_VALUE()` allow us to obtain data from other rows in the same window. `LEAD()` returns values from rows below of the current row; `LAG()` from rows above the current row. For more details, see our article on LEAD vs LAG.

Finally, in the distribution category there are functions like `PERCENT_RANK()` and `CUME_DIST()` that can obtain percentile rankings or cumulative distributions. Check out our Window Functions course for step-by-step instructions on how to use these functions.

Here's an example query with window functions:

```
SELECT
    employee_name,
    department_name,
    salary,
    RANK() OVER (PARTITION BY department ORDER BY salary) position
FROM employee
```

In this query, the window function `RANK()` is used to rank employees by salary. Later in this article, we'll discuss the syntax of the `OVER()` clause and the `PARTITION BY` and `ORDER BY` sub-clauses in detail. For now, we'll only say that they are used to define which records make up the window frame.

Entry-Level Window Function Questions

2. What Is the Syntax of the OVER () Clause ?

The `OVER()` clause is used to define which rows will be in the window frame. The following sub-clauses can be present in the `OVER()` clause:

> `PARTITION BY` defines the partition, or the groups of rows within the window frame, that the window function will use to create a result. (This will be explained below.)
>
> `ORDER BY` defines the order of the rows in the window frame.
>
> `ROWS/RANGE` define the upper and lower limits of the window frame.

All subclauses of `OVER()` are optional and can be omitted. In that case, functions will be performed on the entire window frame.

The following SQL shows the `OVER()` clause at work:

```
SELECT
first_name,
last_name,
department,
salary,
AVG(salary) OVER (PARTITION BY department)
FROM employee
```

For each employee, the query returns their first name, last name, their salary, and the average salary in their department. The clause `OVER (PARTITION BY department)` creates a window of rows for each value in the department column. All the rows with the same value in the department column will belong to the same window. The `AVG()` function is applied to the window: the query computes the average salary in the given department.

The article What Is the OVER Clause? has a complete explanation of the `OVER` clause.

3. Describe the Difference Between Window Functions and Aggregate Functions.

The main difference between window functions and aggregate functions is that **aggregate functions group multiple rows into a single result row**; all the individual rows in the group are collapsed and their individual data is not shown. On the other hand, **window functions produce a result for each individual row**. This result is usually shown as a new column value in every row within the window.

The collapse of rows is an important feature of aggregate functions. For example, we cannot solve the problem "Return all the employees with their salary and the maximum salary in their department" with aggregate functions due to the collapse limitation.

On the similarity side, both aggregate and window functions perform an aggregate-like operation on a set of rows. Some functions like `AVG()`, `MAX()`, `MIN()`, and `SUM()` can be used as both aggregate and

window functions. However, when we need the result of these functions combined with row-level data, it is better to use a window function instead of an aggregate function.

We'll show two SQL queries that return the department name and the max salary of each department. In the first example, we will use `MAX()` as an aggregate function:

```
SELECT  department_name,
        MAX(salary) AS max_salary
FROM   employee
GROUP BY department_name
```

Below, we can see the result of the previous query. Notice that there is one record per department due to the collapsing effect of the `GROUP BY` clause:

department_name	max_salary
Accounting	93000
Sales	134000
Human Resources	78000

In the next example, we'll obtain a similar but slightly different result by using `MAX()` as a window function:

```
SELECT employee_name,
       salary,
       department_name,
       MAX(salary) OVER (PARTITION BY department_name) AS max_salary
FROM  employee
```

As we previously mentioned, window functions do not collapse records. In the following result, we have one row per employee for a total of 5 rows:

employee_name	salary	department_name	max_salary
John Doe	93000	Accounting	93000
Jeremy Smith	134000	Sales	134000
Donna Hayes	120000	Sales	134000
Mark Ron	78000	Human Resources	78000
Denis Serge	72000	Human Resources	78000

Note that we added the columns `employee_name` and salary just by adding their names to the list of columns in the `SELECT`. We could not add them to the query with `GROUP BY` because of the collapse limitation.

In the article SQL Window Functions by Explanation, you can find a detailed explanation of the differences between aggregate and window functions.

4. What's the Difference Between Window Functions and the GROUP BY Clause?

Aggregate functions are frequently used with the `GROUP BY` clause, which defines the groups of rows where the aggregate function will work. The `GROUP BY` clause groups individual rows into sets of rows, allowing the execution of aggregate functions like `SUM()`, `AVG()` or `MAX()` on these sets. Any column from the individual rows cannot be part of the result, as we can see in the following SQL query:

```
SELECT
   department_name,
   AVG(salary)
FROM employee
GROUP BY department_name
```

In the above query, we put only one column in the `SELECT` list: `department_name`. This is possible because the department_name column appears in the GROUP BY clause. However, we cannot add any additional columns in the `SELECT`; only columns specified in the `GROUP BY` are allowed.

The following SQL query is equivalent to the previous one, but it uses window functions instead of `GROUP BY`:

```
SELECT
  department_name,
  AVG(salary) OVER(PARTITION BY department_name)
FROM employee
```

The previous query doesn't have a `GROUP BY` clause because the `AVG()` function is used as a window function. We can recognize that `AVG()` is a window function because of the presence of the `OVER` clause.

I suggest the article SQL Window Functions vs. GROUP BY for a complete comparison between window functions and the `GROUP BY` clause.

5. Show an Example of SQL Window Functions.

This is a good opportunity to mention a query that shows the importance of window functions and at the same time is connected with the queries we showed in previous questions. The query I suggest would solve this task: "Obtain employees' names, salaries, department names, and the average salary of that department."

This query is a simple way to show how we can combine row-level and aggregate data. (The window function returns the aggregate data.)

```
SELECT employee_name,
       salary,
       department_name,
       AVG(salary) OVER (PARTITION BY department) avg_salary
FROM employee
```

Above, we can see the row-level columns `employee_name`, `salary`, and `department` with the average salary of each department, which is calculated by the `AVG()` window function. The `PARTITION BY` sub-clause defines that the windows of records will be created based on the value of the `department_name` column. All records with the same value in `department_name` will be in the same window. The results would look something like this:

employee_name	salary	department_name	avg_salary
John Doe	93000	Accounting	93000
Jeremy Smith	134000	Sales	127000
Donna Hayes	120000	Sales	127000
Mark Ron	78000	Human Resources	75000
Denis Serge	72000	Human Resources	75000

To review more window functions examples, try the article SQL Window Function Examples with Explanations.

Want to learn about window functions? Click here for a great interactive experience!

6. Name Some Common Window Functions.

Window functions can be organized into four categories: aggregate functions, ranking functions, analytic functions, and distribution functions.

The aggregate functions are the regular aggregate functions that you use with GROUP BY: `MAX()`, `MIN()`, `AVG()`, `SUM()`, and `COUNT()`. These functions, as we've already shown, can be used as window functions.

The ranking functions are `ROW_NUMBER()`, `RANK()`, and `DENSE_RANK()`. They are used to obtain different positions in a ranking. You can find a detailed explanation of the ranking functions in the following article.

The analytic functions include `LEAD()`, `LAG()`, `FIRST_VALUE()`, `LAST_VALUE()`, and `NTH_VALUE()`. These functions allow us to obtain data from rows other than the current row (e.g. the previous row, the next row, the last row within a window frame, etc). The `NTILE()` function divides rows within a partition into *n* groups and returns the group number.

Finally the distribution functions `PERCENT_RANK()` and `CUME_DIST()` allow us to obtain data about the percentile or cumulative distribution (respectively) for each row in the window.

I prefer analytic functions because they allow us to compare or calculate the differences between different records within the window (among other things). For example, if I have a time series with stock values, I could calculate how much the stock increased at each moment.

Here's another example of analytic functions. The `LEAD()` and `LAG()` analytical window functions return a column from a subsequent/previous row. So, if we have a table with cryptocurrencies, with a timestamp and a quote value ...

Symbol	Timestamp	Value

Symbol	Timestamp	Value
BTC	2021-05-25 10:30	61400
BTC	2021-05-25 10:40	60300
BTC	2021-05-25 10:50	59800
ETH	2021-05-25 10:30	2700
ETH	2021-05-25 10:40	2750
ETH	2021-05-25 10:50	2820

*Table **Shares***

... we can obtain the following report. To calculate the variation percentage, we need data from two different rows: The value in the current row, and the value in the previous row. The `LEAD()` function will return the value of the previous row. This is the result:

Symbol	Timestamp	Value	% Variation
BTC	2021-05-25 10:30	61400	--
BTC	2021-05-25 10:40	60300	-1.8%
BTC	2021-05-25 10:50	59800	-0.8%
ETH	2021-05-25 10:30	2700	--
ETH	2021-05-25 10:40	2750	1.8%
ETH	2021-05-25 10:50	2820	2.5%

The `% Variation` column was calculated with this type of expression:

(Current_value - Previous_value) / Previous_value

Note that the cryptocurrency value of the previous timestamp can be obtained with:

```
LEAD(value) OVER (PARTITION BY crypto_symbol ORDER BY timestamp)
```

Here's the full query:

```
SELECT Symbol,
       Timestamp,
       Value,
       (Value - LEAD(Value) OVER (PARTITION BY Symbol ORDER BY Timestamp) /
       LEAD(Value) OVER (PARTITION BY Symbol ORDER BY Timestamp) AS "% variation"
FROM  Cryptocurrencies
```

If you want to go deep with `LAG()` and `LEAD()`, I suggest reading the article The LAG() Function and the LEAD() Function in SQL. It has a detailed explanation about how window functions work in ordered windows.

Intermediate Window Function Questions

7. How Do You Define the Window Frame?

Window functions calculate an aggregated result based on a set of records called the "window" or "window frame". Window frames are defined by the `OVER()` clause.

An empty `OVER()` clause means that the window is the entire dataset:

```
SELECT employee_name,
       salary,
       department_name,
       AVG(salary) OVER () avg_salary
FROM employee
```

The above query computes the average salary and displays it next to the other employee details for all employees in the table.

There are several sub-clauses that can be placed inside the `OVER()` clause to precisely define a window.

The `PARTITION BY` sub-clause specifies that all records having the same value in a given column belong to the same window. In other words, `PARTITION BY` specifies how the window is defined. Thus, the following query computes the average salary for each department; the calculations are performed based on groupings of the values in the `department_name` column.

```
SELECT
   employee_name,
   salary,
   department_name,
   AVG(salary) OVER (PARTITION BY department_name) avg_salary
FROM employee
```

`ORDER BY` can also be used inside `OVER()`. It's used to put window rows in a specific order. Ordered windows are very important because they

enable the use of several analytical functions like `LAG()`, `LEAD()`, and `FIRST_VALUE()`.

```
SELECT
   employee_name,
   salary,
   department_name,
   LAG(salary) OVER (ORDER BY salary) prev_salary
FROM employee
```

This query displays the salary of the employee immediately before the current employee in the salary order. Note that you can combine `ORDER BY` and `PARTITION BY` clauses in one query: the ordering is applied to each partition individually.

Two similar `OVER()` sub-clause are `RANGE` and `ROWS`. They define limits for the window frame by putting upper and/or lower limits to the window of records. This means window functions can be calculated based on a subset of rows instead of all the rows in the window. The difference between `ROW` and `RANGE` is explained in detail in our SQL window functions cheat sheet. More on ROWS and RANGE and the different limit options available will be explained in the next two questions.

8. How Does ORDER BY Work with OVER?

Some window functions (like `LAG()`, `LEAD()`, and `FIRST_VALUE()`) work on an ordered window of records. When using one of these functions, we need the `ORDER BY` sub-clause to define the order criteria. A good example of that is the previous query we used to calculate the variation percentage for cryptocurrencies:

```
SELECT Symbol,
       Timestamp,
       Value,
       (Value - LEAD(Value) OVER (PARTITION BY Symbol ORDER BY Timestamp) /
       LEAD(Value) OVER (PARTITION BY Symbol ORDER BY Timestamp) AS “% variation”
FROM  Cryptocurrencies
```

In the above query, the `OVER` clause has two sub-clauses: `PARTITION BY` and `ORDER BY`. `PARTITION BY` defines which records are in each window and `ORDER BY` defines the order of the records in the window. (In this

example, we order records based on their timestamp.) Then the `LEAD()` function returns the value of the previous record.

If the `OVER` clause doesn't include an `ORDER BY` and we don't have ROWS/RANGE, then the window frame is formed by all the rows complying with the PARTITION BY clause. However, when we use an ORDER BY clause without `ROWS/RANGE`, the window frame includes the rows between the first row (based on the ORDER BY clause) and the current row. In other words, those rows that go after the current row will not be included in the window frame. (We'll explain more details on these limits in the next question.)

The window functions that require an `ORDER BY` sub-clause are:

- `RANK()`
- `DENSE_RANK()`
- `LEAD()`
- `LAG()`
- `FIRST_VALUE()`
- `LAST_VALUE()`
- `NTH_VALUE()`
- `PERCENT_RANK()`
- `CUME_LIST()`

To understand more about how `ORDER BY` works, check out the article How to Calculate the Difference Between Two Rows in SQL.

Advanced Window Function Questions

9. Explain What UNBOUNDED PRECEDING Does.

A window frame is a set of rows that are somehow related to the current row, which is evaluated separately within each partition. When we use the `ORDER BY` clause, we can optionally define upper and lower limits for the window frame. The limits can be defined as:

- `UNBOUNDED PRECEDING`
- `n PRECEDING`

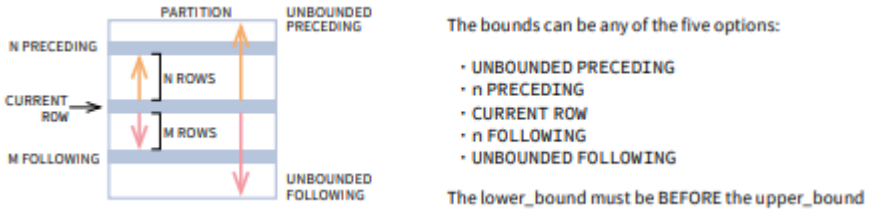
`CURRENT ROW`

`n FOLLOWING`

`UNBOUNDED FOLLOWING`

These limits can be defined with the `RANGE` or `ROWS` sub-clauses in the `OVER()` clause. `UNBOUNDED PRECEDING` indicates the lower limit of the window is the first record in the window; in the same way, the upper limit can be defined with `UNBOUNDED FOLLOWING` or `CURRENT ROW`. These limits should be used only with ordered windows.

In the following image, we can see how different limits function:



For example, if we want to obtain the average value of a cryptocurrency considering only the values occurring up to the current value, we can use the following `OVER()` clause:

```
AVG(value) OVER (PARTITION BY symbol_name
                 ORDER BY timestamp
                 RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
                 )
```

In this `OVER()` clause, we defined the `CURRENT ROW` as the upper limit of the window to calculate the average. This is exactly what we need, as we don't want to include values registered after the current timestamp in the average calculation.

10. Describe SQL's Order of Operations and Window Functions' Place in This Order.

The sub-clauses of an SQL `SELECT` are executed in the following order:

FROM / JOINS

WHERE

GROUP BY

Aggregate Functions

HAVING

Window Functions

SELECT

DISTINCT

UNION / INTERSECT / EXCEPT

ORDER BY

OFFSET

LIMIT / FETCH / TOP

Because window functions are calculated during step 6, we cannot put them in the `WHERE` clause (which is calculated in step 2). However, we can bypass this limitation by using a CTE (common table expression), where we can call window functions and store their results as columns in the CTE. The CTE will be treated like a table and the window function results will be evaluated as regular column values by the `WHERE`.

There is an interesting article on why window functions are not allowed in WHERE clauses that you should read if you're looking for some examples.

On the other hand, we can use aggregation/grouping results in window functions, as they are already computed by the time the window functions are processed.

Want To Enhance Your SQL Window Functions Skills?

This article covers several possible job interview questions about SQL window functions. My final advice is about connecting the questions in this article with the questions you'll get during an interview. Here it is:

Try to associate each question in this article with a window function topic, like "`OVER` clause", "name a function" or "`ORDER BY` sub-clause". Then, if you are asked about window functions during the interview, identify the topic of the question and use the information here to discuss the topic.

Check out our Window Functions course. 218 interactive exercises.