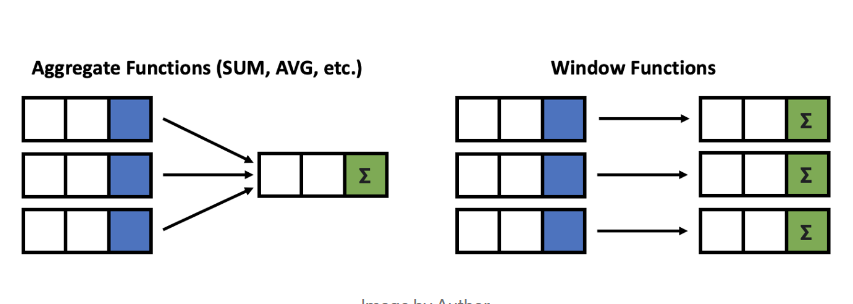
**What are window functions?**

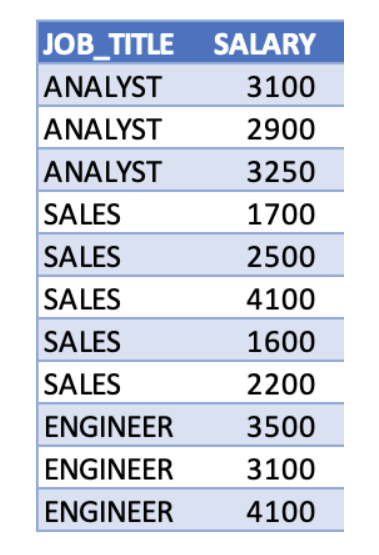
Window functions are similar to the aggregation done in the GROUP BY clause. However, rows are not grouped into a single row, each row retains their separate identity. That is, a window function may return a single value for each row. Here’s a good visualization of what I mean by that.

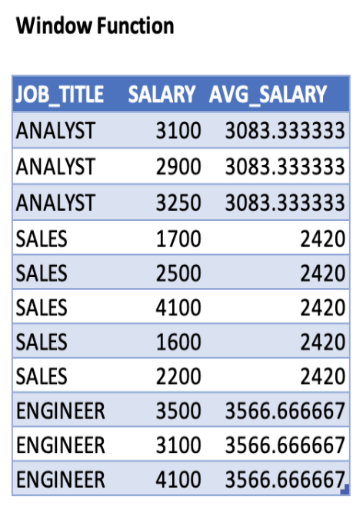
****

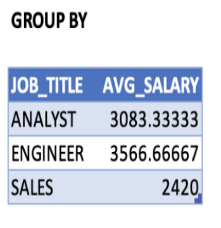
Notice how the GROUP BY aggregation on the left-hand side of the picture groups the three rows into one single row. The window function on the right-hand side of the picture is able to output each row with an aggregation value. This may save you from having to do a join after the GROUP BY.

**Example: Group By versus Window Function**

Here’s quick example to give you a taste of what a window function dose.

Let’s say we have some salary data and we want to find to create a column that gives us the average salary for each job title.





# Why use Window Functions?

One major advantage of window functions is that it allows you to work with both aggregate and non-aggregate values all at once because the rows are not collapsed together.

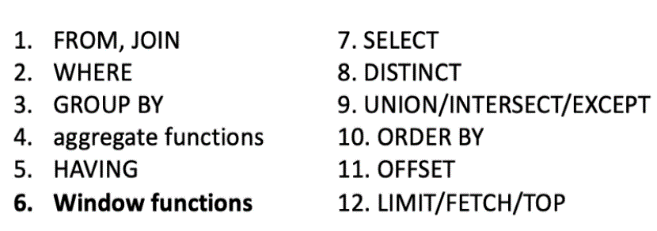
Window functions are also simple to use and read. That is, they can reduce the complexity of your queries, which makes it easier to maintain down the road.

In addition, they can help with performance issues. For example, you can use a window function instead of having to do a self-join or cross-join.

I promise, window functions are truly amazing and great to know.

**Important Note:**

Before we start, it is important to note that in terms of the order of operations in SQL, window functions come in sixth on the list.



This is important because based off of this logical order, window functions are allowed in SELECT and ORDER BY, but they are not allowed in FROM, WHERE, GROUP BY, or HAVING clauses.

**Note:** If you really need to have it inside a WHERE clause or GROUP BY clause, you may get around this limitation by using a subquery or a WITH query.

**Window Function Syntax**

Here’s what the generic syntax looks like for a window function in the SELECT clause.



There’s a lot of words here, so let’s look at some definitions:

* **window\_function**is the name of the window function we want to use; for example, sum, avg, or row\_number (we’ll learn more about these later)
* **expression**is the name of the column that we want the window function operated on. This may not be necessary depending on what window\_function is used
* **OVER** is just to signify that this is a window function
* **PARTITION BY**divides the rows into partitions so we can specify which rows to use to compute the window function
* **partition\_list**is the name of the column(s) we want to partition by
* **ORDER BY**is used so that we can order the rows within each partition. This is optional and does not have to be specified
* **order\_list**is the name of the column(s) we want to order by
* **ROWS** can be used if we want to further limit the rows within our partition. This is optional and usually not used
* **frame\_clause** defines how much to offset from our current row

Don’t worry about memorizing the definitions and syntax or even fully understanding what it means exactly right now. Everything will make a lot more sense once you look at the examples in the article and get an intuitive understanding of how to go about writing a window function.

**List of window functions**

1.Over ()

2.Partitions ()

**Ranking Window Function**

1. Rank () and Dense Rank ()

2. Row\_Number()

3. Ntile(N)

**Value Window Functions**

1. Lag and Lead

2. First\_Value (), Last\_Value ()