SQL for Citizen Data Scientists

SQL Aggregate Functions - Learn By Example

SETScholars & WACAMLDS

It is often necessary to summarize data for analysis and reporting purposes. Be it determining the number of rows in a table, obtaining the sum of column's values, or finding the column's highest, lowest, or average value.

Aggregate functions are used for this type of retrieval. Although each DBMS has its own set of aggregate functions, the common aggregate functions implemented by all major DBMSs include:

Functions	Description
Count()	Returns the number of values in a column
Sum()	Returns the sum of the values in a column
Avg()	Returns the average value of a column
Min()	Returns the lowest value in a column
Max()	Returns the highest value in a column

These functions are so efficient that they usually return results much faster than you can compute them within your client application.

Sample Table

To help you better understand the examples, and enable you to follow along with the tutorial, we are going to use the following sample table.

This table is part of an 'Employee Management System' that contains basic information about employees.

ID	Name	Age	Job	Salary	Email
1	Joe	28	Manager	60000	joe@mail.com
2	Eve	24	Developer	32000	eve@mail.com
3	Sam	26	Janitor	9000	*NULL*
4	Kim	25	Manager	55000	kim@mail.com
5	Bob	23	Developer	30000	bob@mail.com
6	Max	27	Janitor	10000	*NULL*

The AVG() Function

`AVG()` function is used to calculate the average value of a specific column. `AVG()` requires that the column name must be specified as a function parameter.

This first example uses `AVG()` to return the average salary of all employees in the 'Employees' table:

```
SELECT AVG(Salary) AS avg_salary
FROM Employees;
```

The SELECT statement above returns a single value containing the average salary of all employees. `avg_salary` is an alias for the newly computed column.

With `AVG()` you can also calculate the average value of specific rows. The following example returns the average salary of all 'Managers':

```
SELECT AVG(Salary) AS avg_salary
FROM Employees
WHERE Job='Manager';
```

This example is the same as the previous SELECT statement, but this time it contains a WHERE clause. The WHERE clause filters only employees with a manager's job title, and therefore, the value returned in `avg_salary` is the average of the salaries of those employees only.

The MAX() Function

`MAX()` returns the highest value in a specified column. Just like `AVG()`, `MAX()` also requires that the column name be specified as a function parameter.

Here `MAX()` returns the salary of the highest paid employee in the 'Employees' table.

```
SELECT MAX(Salary) AS max_salary
FROM Employees;
```

Although `MAX()` is commonly used to find the highest value in a numeric column, many DBMS allow it to be used to return the highest value in a text column. When used with text data, `MAX()` returns the highest value in alphabetical order.

```
SELECT MAX(Name)
FROM Employees;
```

The MIN() Function

`MIN()` does the exact opposite of `MAX()`; it returns the lowest value in a specified column.

Here `MIN()` returns the salary of the lowest paid employee in the 'Employees' table.

```
SELECT MIN(Salary) AS min_salary
FROM Employees;
```

Like `MAX()`, `MIN()` is commonly used to find the lowest value in a numeric column. When used with text data, however, `MIN()` returns the lowest value in alphabetical order.

```
SELECT MIN(Name)
FROM Employees;
```

The SUM() Function

`SUM()` is used to calculate the sum of the values in a specific column. Like other aggregate functions, `SUM()` also requires that the column name be specified as a function parameter.

The following example returns the total of the salary (the sum of all the Salary values) paid to the employees.

```
SELECT SUM(Salary) AS total_salary
FROM Employees;
```

The COUNT() Function

`COUNT()`, as its name suggests, is used to count the number of rows in a table or the number of rows that match a specific criterion.

Unlike others, you can pass an asterisk or a column name to `COUNT()` as a function parameter. Depending on what you pass, `COUNT()` works differently:

1. `COUNT(*)` : If you pass an asterisk, it counts the number of rows in the table, whether the column contains NULL values or not.
2. `COUNT(column)` : If you pass a column name instead of an asterisk, it counts the number of values in that column, ignoring the NULL values.

The first example returns the total number of employees in the 'Employees' table:

```
SELECT COUNT(*) AS num_emp
FROM Employees;
```

In this example, an asterisk is passed to the `COUNT()`, so it counts the number of rows in the table even though some columns have NULL values.

The next example counts just the employees with an e-mail address:

```
SELECT COUNT(Email) AS num_emp
FROM Employees;
```

In this example, the column name 'Email' is passed to the `COUNT()` function, so it only counts non-NULL values in that column. Since only 4 out of 6 employees have email addresses, `num_emp` is 4.

Using Expressions

In addition to using a column name, you can also use an expression as an argument to aggregate functions.

For example, you may want to find the total of salary when each employee gets a 10% increase. You can achieve this through the following query:

```
SELECT SUM(Salary * 1.1) AS total_salary
FROM Employees;
```

While this example uses a simple expression, expressions used as arguments to aggregate functions can be as complex as necessary as long as they return a number, string, or date.

Aggregates on Distinct Values

When using aggregate functions, you have a choice of performing calculations on all values in a column or only on unique values. To only include unique values, you need to specify the DISTINCT argument.

The following example uses the `COUNT()` function to return the count of unique jobs.

```
SELECT COUNT(DISTINCT Job) AS jobs
FROM Employees;
```

Here the DISTINCT keyword makes sure that the COUNT only takes into account unique jobs.

Note that the DISTINCT keyword can only be used when you specify a column name in the `COUNT()` function and not with `COUNT(*)`.

Combining Aggregate Functions

So far we have been using only one aggregate function in our SELECT statement, but in fact you can use as many aggregate functions as you need.

Here is a query that uses all of the common aggregate functions to return five values (the number of employees, and the highest, lowest, average and total of their salary)

```
SELECT COUNT(*) AS num_emp,
      MIN(Salary) AS min_salary,
      MAX(Salary) AS max_salary,
      AVG(Salary) AS avg_salary,
         SUM(Salary) AS total_salary
FROM Employees;
```

num_emp	min_salary	max_salary	avg_salary	total_salary
6	9000	60000	32666	196000

Aggregates on Grouped Data

So far we have executed aggregate functions on all the data in a table. But, what if you want to execute them separately for each job? To answer such a query, you will need to divide the data into logical sets by grouping them and then perform aggregate calculations on each group.

With the GROUP BY clause you can specify explicitly how the data should be grouped and then work on each group separately.

To demonstrate, let's extend the previous query to execute the same five aggregate functions separately for each job.

```
SELECT Job,
      COUNT(*) AS num_emp,
      MIN(Salary) AS min_salary,
      MAX(Salary) AS max_salary,
      AVG(Salary) AS avg_salary,
         SUM(Salary) AS total_salary
FROM Employees
GROUP BY Job;
```

Job	num_emp	min_salary	max_salary	avg_salary	total_salary
Developer	2	30000	32000	31000	62000
Janitor	2	9000	10000	9500	19000
Manager	2	55000	60000	57500	115000

Now that the GROUP BY clause is included, SQL grouped the rows of the same values present in the 'Job' column together and then applied five aggregate functions to each of the three groups.

Read in detail about the Group By clause here.

How Nulls Are Handled

Before performing aggregations, it is important to know how these five aggregate functions handle NULL values in a column.

To demonstrate let's consider the following 'Grocery' table:

ID	Name	Qty
1	Oranges	5
2	Apples	8
3	Bananas	6
4	Lettuce	7
5	Tomatoes	5
6	Eggs	12

Now let's perform five aggregate functions on the 'Qty' column:

```
SELECT COUNT(*) AS num_rows,
      COUNT(Qty) AS num_vals,
      MIN(Qty) AS min_val,
      MAX(Qty) AS max_val,
      AVG(Qty) AS avg_val,
         SUM(Qty) AS total
FROM Grocery;
```

num_rows	num_vals	min_val	max_val	avg_val	total
6	6	5	12	7	43

The results are as you would expect: both `count(*)` and `count(Qty)` return the value 6, `min(Qty)` returns the value 5, `max(Qty)` returns 12, `avg(Qty)` returns 7, and `sum(Qty)` returns 43.

Next, let's add a NULL value to the 'Qty' column:

ID	Name	Qty
1	Oranges	5
2	Apples	8
3	Bananas	6
4	Lettuce	7
5	Tomatoes	5
6	Eggs	12
7	Milk	*NULL*

If you run the same query again, you will get the following result:

num_rows	num_vals	min_val	max_val	avg_val	total
7	6	5	12	7	43

As you can see even with the addition of the NULL value for the table, the `sum()`, `min()`, `max()`, and `avg()` functions all return the same value, indicating that they ignore any NULL value.

The `count(*)`, however, now returns the value 7, which is valid since there are seven rows in the 'grocery' table, while the `count(Qty)` still returns the value 6. The difference is that `count(*)` counts the number of rows, while `count(Qty)` counts the number of values contained in the 'Qty' column and ignores null values.