2022-06-07T18:00:00+02:00
Tihomir Babic

How to Combine Two Aggregate Functions in SQL

Having trouble using two aggregate functions in one query? This article will show you how to do it the right way – actually, the two right ways.

In data analysis and reporting, we often need to count the number of records or sum them up and then calculate the average of this count or sum.

Translated to SQL logic, this is the aggregation of aggregated data, or multi-level aggregation. For aggregation purposes, there are the SQL aggregate functions. And for multi-level aggregation, you'd use (at least) two aggregate functions at the same time.

If you're interested in quality reporting, you'll need much more than SQL's aggregate functions. However, they're certainly the basis of good reporting. The best way to build on that basis is to take our interactive Creating Basic SQL Reports course. It offers a whole section dedicated solely to computing multi-level aggregations in SQL and summarizing data using aggregate functions. It also covers data classification using the `CASE WHEN` and `GROUP BY`, calculating multiple metrics in one report, and comparing groups within the same report. With 97 interactive exercises, this course will help you master reporting in SQL!

LearnSQL.com provides a comprehensive learning experience when it comes to learning SQL. With 30 interactive courses at various levels, the platform will help you to learn the skills you need to tackle reporting in SQL with confidence.

To make it even easier, I'll show you here how to combine two aggregate functions in SQL. If your knowledge of aggregate functions is rusty, this guide to SQL aggregate functions can help you follow this article. You can also practice on these aggregate functions examples.

Sample Data

We have a table called `new_users`. It collects data about the new users of an app in the South American market. The columns are:

- `id` – The ID for each individual record.
- `date` – The date when the users joined.
- `number_of_new_users` – The number of new users by date.
- `city` – The users' city.
- `country` – The users' location country.

You see below there are two dates in total, with every date having a different number of users from two countries and two cities in each country.

id	date	number_of_new_users	city	country
1	2022-05-10	29	Cordoba	Argentina
2	2022-05-10	47	Buenos Aires	Argentina
3	2022-05-10	22	Bogotá	Colombia
4	2022-05-10	52	Medellín	Colombia
5	2022-05-11	37	Cordoba	Argentina
6	2022-05-11	19	Buenos Aires	Argentina
7	2022-05-11	41	Bogotá	Colombia
8	2022-05-11	87	Medellín	Colombia

How would you use the `AVG()` and `SUM()` functions to calculate the average daily number of new users by country?

The Naive Solution

The logic behind the naive approach is, actually, quite sound. It's a shame that this naive solution is no solution at all. You'll soon realize why.

If you only think about the mathematical approach, it's simple: you first have to sum the number of new users per country every day, then calculate the average of that sum.

Transferred to SQL functions, it seems logical to write something like this:

```
SELECT country,
     AVG(SUM(number_of_new_users)) AS average_new_daily_users
FROM new_users
GROUP BY country;
```

Why not, right? There's a very simple reason why not: SQL does not allow the nesting of aggregate functions. In other words, you can't use an aggregate function within an aggregate function. Well, you can, but the query will return an error saying exactly what I just said:

Data Output Explain Messages Notifications

```
ERROR:  aggregate function calls cannot be nested
LINE 2:      AVG(SUM(number_of_new_users)) AS average_new_daily_user...
                 ^
SQL state: 42803
Character: 26
```

How do you fix this query to combine two aggregate functions in SQL?

There are two options: using a subquery or using Common Table Expressions (CTEs).

Real Solution 1: Subquery

The first option is to combine two aggregate functions using a subquery. The subquery is a query within the main query. When creating reports, they are usually found in the `SELECT`, `FROM`, or `WHERE` clauses.

In this example, I'll put the subquery in the `FROM` clause.

```
SELECT country,
     AVG(ds.sum_new_users) AS average_daily_new_users
FROM (SELECT date,
          country,
          SUM(number_of_new_users) AS sum_new_users
```

```
      FROM new_users
      GROUP BY date, country) AS ds
GROUP BY country;
```

The principle when combining two aggregate functions is to use the subquery for calculating the 'inner' statistic. Then the result is used in the aggregate functions of the outer query.

LearnSQL.com is an online platform designed to help you master SQL. It offers 30 interactive courses that range in difficulty from beginner to advanced. Each course delivers both theoretical knowledge and hands-on exercises so that you can solidify these new ideas.

The above code selects the date and country and calculates the sum of the column `number_of_new_users`. This returns the total daily number of new users. I gave the subquery the alias ds, which is short for 'daily sum'.

Once I got the daily sum, I referenced this in the outer query by calculating the average of the column `ds.sum_new_users` – i.e. the average of the daily new users. I want this average to be by country; that's why I grouped the data by the country column.

country	average_daily_new_users
Colombia	101
Argentina	66

Real Solution 2: CTE

The other option for combining aggregate functions in SQL is using a CTE instead of a subquery. A CTE is a tidier and "closer to the mathematical logic" version of a subquery. It is an expression that allows you to create a temporary result, which you can reference in another `SELECT` statement. You can use the result of a CTE as you would any other table. The difference is that the CTE result exists only when a CTE is run together with the query using the CTE. A more detailed explanation of the CTEs can be found in the article What Is a CTE?.

The query below will give you the same result as the subquery solution:

```
WITH ds AS (
  SELECT date,
    country,
          SUM(number_of_new_users) AS sum_new_users
  FROM new_users
  GROUP BY date, country)

SELECT country,
       AVG(ds.sum_new_users) AS average_daily_new_users
FROM ds
GROUP BY country;
```

Every CTE is introduced using the `WITH` keyword. The same is true for my CTE named `ds`. After the keyword `AS` comes the CTE definition. In this case, it's the same `SELECT` statement as in the subquery from the previous example. Here comes the `sum()`: it will, again, return the sum of new users by country and date.

The second SELECT statement references CTE and calculates the average of the sum returned by CTE.

Unlike nesting the subquery, using the CTE allows you to use the aggregate functions in the logical order: first `SUM()`, then `AVG()`. The 'inner' function is used in the CTE, while the second `SELECT` statement is for the 'outer' aggregate function.

The result will be the same as with the subquery; no need for me to show it again. However, it's always good to learn more about the differences between a subquery and a CTE and then practice using CTEs on real-life examples.

Become a Master of Data Aggregation and Reporting

The problem of using two aggregate functions in SQL is ubiquitous. The more complex reporting becomes, the more you'll use multi-level aggregations. You now know subqueries and CTEs are the two solutions for that.

LearnSQL.com is an online platform designed to help you master SQL. LearnSQL.com allows you to choose from a full learning track, mini-tracks to sharpen targeted skills, and individual courses. You can also