Calculations in SELECT Statements

by Sophia

WHAT'S COVERED

This lesson explores using calculations in SELECT statements to create more complex data results, in two parts. Specifically, this lesson will cover:

1. How Calculations Aid in Data Analysis and Reporting
2. Using a Calculation in a SELECT Statement

1. How Calculations Aid in Data Analysis and Reporting

Calculations in SELECT statements are important because they enable you to transform, manipulate, and derive meaningful insights from your data directly within your SQL queries. They are a powerful tool for data analysis, reporting, and presentation. Here's why calculations in SELECT statements are crucial:

BIG IDEA

By transforming and deriving data with calculations, you can perform various mathematical, logical, or string operations. You can create new columns representing aggregated values, percentages, averages, differences, etc. This transformation converts raw data into a more meaningful and interpretable format, facilitating decision making.

You can get more focused customized reporting by performing calculations within the SELECT statement; you can generate customized reports tailored to your specific needs. This flexibility allows you to present data in a format that suits your requirements, whether it's summarizing sales data, calculating growth rates, or computing ratios. Performing calculations directly in the database via SQL can be more efficient than exporting and processing data externally. Database engines are optimized for such computations, potentially resulting in faster query execution times.

Calculations enable you to gain real-time insights into your data without preprocessing it outside the database. This is especially valuable when working with large datasets or performing frequent and iterative analyses. Calculations within SQL queries maintain data integrity by ensuring that all calculations are consistent and accurate. It reduces the risk of errors that might occur if data is manipulated in separate external tools. Calculations can be combined with joins to create more complex analyses that involve data from multiple tables. This is especially useful when working with relational databases. Calculations are often used in combination with aggregation functions and grouping to summarize data at different levels and produce meaningful insights for different segments.

KEY CONCEPT

In effect, calculations in SELECT statements empower you to extract actionable insights directly from your data within the database environment. They save time, enhance data accuracy, and contribute to better decision making by providing you with the tools to analyze and present data in ways that address your specific business needs.

In most cases, the computed values are not stored in the database. Rather, these values are typically calculated at the time of the query so that they reflect the most up-to-date data.

```
SELECT invoice_id, SUM(quantity * unit_price)
FROM invoice_line
GROUP BY invoice_id;
```

EXAMPLE If we wanted to calculate the total amount by invoice_id in the invoice_line table, we would need a query like the following: This query would calculate the total of the invoices by adding up the product of the quantity multiplied by the unit_price per item. In this particular example, we already have that information available in the total column of the invoice table, but in the invoice table, the value is rounded up to the nearest integer, whereas here in the calculated results, it appears with two decimal places.

Query Results

Row count: 412

invoice_id	sum
384	0.99
351	1.98
184	3.96
116	8.91
87	6.94
273	1.98
394	3.96
51	3.96
272	0.99

2. Using a Calculation in a SELECT Statement

We can also do the same for a more complex query with dates to find the employee's age when they were hired, based on their birthdate. At a very basic level, we could simply use a minus sign to subtract the birth_date from the hire_date, like this:

```
SELECT employee_id, hire_date, birth_date, hire_date - birth_date
FROM employee;
```

Query Results

Row count: 8

employee_id	hire_date	birth_date	{ "days": 14787 }
1	2002-08-14T00:00:00.000Z	1962-02-18T00:00:00.000Z	{ "days": 14787 }
2	2002-05-01T00:00:00.000Z	1958-12-08T00:00:00.000Z	{ "days": 15850 }
3	2002-04-01T00:00:00.000Z	1973-08-29T00:00:00.000Z	{ "days": 10442 }
4	2003-05-03T00:00:00.000Z	1947-09-19T00:00:00.000Z	{ "days": 20315 }
5	2003-10-17T00:00:00.000Z	1965-03-03T00:00:00.000Z	{ "days": 14107 }
6	2003-10-17T00:00:00.000Z	1973-07-01T00:00:00.000Z	{ "days": 11065 }
7	2004-01-02T00:00:00.000Z	1970-05-29T00:00:00.000Z	{ "days": 12271 }
8	2004-03-04T00:00:00.000Z	1968-01-09T00:00:00.000Z	{ "days": 13204 }

However, the result is based on days, as shown above. To convert this to a year, we would have to do some further calculations. First, we will need to pull out just the number of days by using the date_part function, like this:

```
SELECT employee_id, hire_date, birth_date, date_part('day',hire_date - birth_date)
FROM employee;
```

Query Results

Row count: 8

employee_id	hire_date	birth_date	date_part
1	2002-08-14T00:00:00.000Z	1962-02-18T00:00:00.000Z	14787
2	2002-05-01T00:00:00.000Z	1958-12-08T00:00:00.000Z	15850
3	2002-04-01T00:00:00.000Z	1973-08-29T00:00:00.000Z	10442
4	2003-05-03T00:00:00.000Z	1947-09-19T00:00:00.000Z	20315
5	2003-10-17T00:00:00.000Z	1965-03-03T00:00:00.000Z	14107
6	2003-10-17T00:00:00.000Z	1973-07-01T00:00:00.000Z	11065
7	2004-01-02T00:00:00.000Z	1970-05-29T00:00:00.000Z	12271
8	2004-03-04T00:00:00.000Z	1968-01-09T00:00:00.000Z	13204

Next, we need to convert the days to years by dividing the age in days by 365.25 (with the .25 used to account for leap years). However, if we simply divide by 365.25, the result is still displayed in days instead of years:

```
SELECT employee_id, hire_date, birth_date, (hire_date - birth_date)/365.25
FROM employee;
```

Query Results

Row count: 8

employee_id	hire_date	birth_date	{ "days": 40, "hours": 11, "minutes": 37, "seconds": 49, "milliseconds": 404.517 }
1	2002-08-14T00:00:00.000Z	1962-02-18T00:00:00.000Z	{ "days": 40, "hours": 11, "minutes": 37, "seconds": 49, "milliseconds": 404.517 }
2	2002-05-01T00:00:00.000Z	1958-12-08T00:00:00.000Z	{ "days": 43, "hours": 9, "minutes": 28, "seconds": 42, "milliseconds": 381.93 }
3	2002-04-01T00:00:00.000Z	1973-08-29T00:00:00.000Z	{ "days": 28, "hours": 14, "minutes": 7, "seconds": 38, "milliseconds": 316.222 }
4	2003-05-03T00:00:00.000Z	1947-09-19T00:00:00.000Z	{ "days": 55, "hours": 14, "minutes": 51, "seconds": 59, "milliseconds": 507.187 }
5	2003-10-17T00:00:00.000Z	1965-03-03T00:00:00.000Z	{ "days": 38, "hours": 14, "minutes": 56, "seconds": 55, "milliseconds": 195.072 }
6	2003-10-17T00:00:00.000Z	1973-07-01T00:00:00.000Z	{ "days": 30, "hours": 7, "minutes": 3, "seconds": 49, "milliseconds": 158.111 }
7	2004-01-02T00:00:00.000Z	1970-05-29T00:00:00.000Z	{ "days": 33, "hours": 14, "minutes": 18, "seconds": 28, "milliseconds": 829.569 }
8	2004-03-04T00:00:00.000Z	1968-01-09T00:00:00.000Z	{ "days": 36, "hours": 3, "minutes": 36, "seconds": 50, "milliseconds": 266.94 }

Instead, let's divide the number of days by 365.25 and then round it to the nearest integer:

```
SELECT employee_id, hire_date, birth_date, ROUND(date_part('day',hire_date - birth_date)/365.25)
FROM employee;
```

Query Results

Row count: 8

employee_id	hire_date	birth_date	round
1	2002-08-14T00:00:00.000Z	1962-02-18T00:00:00.000Z	40
2	2002-05-01T00:00:00.000Z	1958-12-08T00:00:00.000Z	43
3	2002-04-01T00:00:00.000Z	1973-08-29T00:00:00.000Z	29
4	2003-05-03T00:00:00.000Z	1947-09-19T00:00:00.000Z	56
5	2003-10-17T00:00:00.000Z	1965-03-03T00:00:00.000Z	39
6	2003-10-17T00:00:00.000Z	1973-07-01T00:00:00.000Z	30
7	2004-01-02T00:00:00.000Z	1970-05-29T00:00:00.000Z	34
8	2004-03-04T00:00:00.000Z	1968-01-09T00:00:00.000Z	36

This is a great example of calculation being used. You could even convert the hire_date to use now() to get the current date to find out the employee's current age at the time that the query is run:

```
SELECT employee_id, hire_date, birth_date, ROUND(date_part('day',now() - birth_date)/365.25)
FROM employee;
```

Give it a try and see what happens. This is not a calculation you could easily store in the database, as the value would constantly change. If it were to be stored, you would have to update the table daily. Consider that aspect of live data as we look ahead to using views.

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then, enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

In this lesson, you learned **how calculations aid in data analysis and reporting**—more specifically, that calculations used in SELECT statements enable data transformation, analysis, and customization. In these calculations, various mathematical, logical, and string functions are applied to data retrieved from a database. This leads to valuable insights that can be gained from the derived values. You learned that they can also be used to keep queries current. Calculations allow for data transformation, enabling aggregation, computation, and transformation of values to produce new columns. Business performance can be comprehensively understood by calculating total sales, average prices, or growth rates. You learned that calculations help tailor data presentation to meet specific needs, facilitating customized reporting. This flexibility is essential for generating insightful reports that address specific questions or provide insights into trends and patterns.

Data analysis is made more efficient by **using calculations within SELECT statements**. This avoids the time-consuming process of exporting data and external processing by processing it directly within the database. With real-time processing, even large datasets can be analyzed quickly and iteratively. A SELECT statement enables data transformation, custom reporting, and efficient real-time insights, ultimately facilitating informed decision making and a deeper understanding of data-driven narratives.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND FAITHE WEMPEN (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR **TERMS OF USE**.