# Calculations in SELECT Statements

*by Sophia*

WHAT'S COVERED

This tutorial explores using calculations in SELECT statements to create more complex data results in two parts:

1. Keeping Queries Current
2. Calculating Age Example

## 1. Keeping Queries Current

We have covered some basic calculations in prior tutorials, which can be quite useful to create additional computed columns through expressions and formulas. 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.

For 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:

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

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 |

This query would calculate the total of the invoices by adding up the product of the quantity multiplied by the unit_price per item. Although in this case, the value is also stored in the invoice table in the total column, the value in the invoice table is rounded up to the nearest integer.

---

# 2. Calculating Age Example

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. If we simply calculated the difference between the two dates:

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 } |

The result is based on the days. To convert this to a year, we would have to divide it by the number of days in a year. To include the potential leap years, we could divide the age by 365.25 after we've converted the number of days to a number, and pulled out just the number with the date_part function:

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 |

If we simply divided the value by 365.25 without the conversion, the result would be 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 be changing. If it were to be stored, you would have to update the table every day. Consider that aspect of live data as we look ahead to the use of views.

---

## Video Transcription

[MUSIC PLAYING] We have the ability to be able to utilize and do calculations to be able to name new columns based on actual life data. This way, we don't have to store those values in any location. We can just automatically calculate it based on the underlying raw data.

For example, in this case here, we're going to select from the invoice line, showing the sum of the quantity multiplied by the unit price, grouped by the invoice ID. This will provide us with all the information, all calculated in real time. So if there are any changes with the data, it'll automatically get updated.

Know, too, that we can also utilize aliases for that particular calculation, so that we're not just left with the word "sum" as part of a name. So for example, in this case here, of how to remove the alias, it would just show the calculation. It doesn't really identify exactly what that component is going to be reflecting.

You can do this with any column, any components at all, to be able to create those calculations to make use of them at a later date.

[MUSIC PLAYING]

[MUSIC STOPS]

### TRY IT

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.

### SUMMARY

We can have live calculations of data in the SELECT statements to simplify data.

Source: Authored by Vincent Tran