Theory: Expressions

12 minutes 0 / 5 problems solved

You already know quite a bit about data types and literals in SQL; it's time to dig deeper and discuss arithmetic, logic, and string expressions. As a cool bonus, we'll teach you how to make a simple calculator with this knowledge!

§1. Arithmetic

If you know a bit of school math, you're already familiar with arithmetic expressions in SQL; look at some examples: $2 + 2$, $2 + 2/2$, and $(2 - 2) * 2$.

The basic set of arithmetic operators supported in SQL is the following:

- - (unary minus that changes the sign of value)
- * (multiplication), / (division), % (modulo that returns remainder of integer division),
- + (addition), - (subtraction)

SQL supports the common rules of operator precedence; in the list above operators are sorted by the decrease of priority. It also supports brackets to make an operator take priority over any other.

You can also utilize brackets to improve code readability even if you do not need them for the correct evaluation of an expression. Compare these: -2+2*2-2/2 and (-2)+(2*2)-(2/2). The second one is much easier to read!

§2. String expressions

SQL supports a variety of functions to process text values: let's discuss some of them.

- Function `concat(s_l, s_r)` returns a concatenation of strings *s_l* and *s_r* as a result. For example, `concat('Hyper', 'skill')` returns 'Hyperskill'.
- Function `char_length(s)` can be applied to count the number of characters in argument *s*. For instance, `char_length('apple')` equals 5.
- To reverse a string you can call the function with quite a self-explanatory name – `reverse(s)`. For example, `reverse('Madam, I''m Adam')` returns 'madA m''I ,madaM'.
- Function `replace(s_in, s_what, s_with)` returns a string *s_in,* where all occurrences of substring *s_what* are replaced by string *s_with*. For example, `replace('abracadabra', 'bra', '')` returns 'acada'.
- To cut a substring of *n_char* symbols starting from position *i_from* in string *s,* you can use the function `substr(s, i_from, n_char)`. For example, `substr('Hello, data!', 8, 4)` returns 'data'.

SQL dialects differ a lot when it comes to string processing. We provided syntax for PostgreSQL; in your projects, please refer to the documentation from your data management system vendor.

§3. Logic expressions

For logic values, SQL supports operators from boolean algebra: NOT, AND, and OR (sorted here by the decrease of priority). You can write a complex logic expression like those in arithmetics, for instance, expression `((true AND true) AND (NOT false)) OR false` that can be evaluated as true.

§4. Comparisons

There is another important class of expressions – **comparisons**. SQL supports the following set of comparison operators:

Current topic:

Expressions

Topic depends on:

× Literals

Topic is required for:

Basic SELECT statement

The NULL value

Basic UPDATE statement

Table of contents:

- = (equality check)
- < (less), > (greater)
- <= (less or equal), >= (greater or equal)
- <> (not equal).

You can compare numeric or string values according to lexicographical order (the order of the letters in the English alphabet); for example, comparisons $2 < 3$, $2 + 2 = 4$, char_length('') = 0 will return TRUE, while 'ab' = 'ba', 'a' >= 'aa' will return FALSE.

The result of a comparison expression is a truth value that can be part of a logic expression, for instance, (char_length('hyperskill') = 5 + 5) AND (concat('hyper','skill') = 'hyperskill'). So you can combine different types of data in one expression if needed!

§5. Calculator

After reading this topic, you can now utilize SQL when you do not have a calculator at hand! This sounds like quite a lifehack. Moreover, you know how to evaluate truth expressions and process strings. Very impressive!

In SQL you can select not only a literal but an expression as well. Let's provide a template for a simple SQL query that extracts an expression:

```
1 SELECT expression;
```

The statement consists of three parts: the keyword SELECT, the expression that we want to evaluate (here you can place any correctly specified expression), and a semicolon that defines the end of the query.

For example, the code below evaluates the expression $(2 + 2) * 15$.

```
1 SELECT (2+2)*15;
```

The query evaluation result is 60.

Well, this covers a lot. Shall we now move on to practice?

Report a typo

319 users liked this theory. **2** didn't like it. **What about you?**

Start practicing

Comments (14) Hints (0) Useful links (0) Show discussion