

SQL standard RegEx operators, functions, syntax

1. RegEx 101 - Recap

Regular expressions (or RegEx for short) are a way to describe patterns in text. They allow us to search, match or manipulate a given string using flexible rules rather than just searching based on a fixed sequence of characters. They allow us to achieve complex tasks such as searching for all email addresses in a text document (which we will show you how to do later on) or extracting text data in a certain format from a database.

How do they work?

When you apply a valid (syntax-wise) regular expression to a string, a regex engine scans through the text and checks whether or not it can find a sequence of characters that fits all the rules of the pattern you've defined

What is a regex engine?

A regex engine is a part of a program or library that reads, interprets and executes these patterns (regular expressions). In SQL, different database systems use slightly different regex engines, which affects the syntax and supported functions - we'll look into this a bit later on in our presentation.

2. RegEx syntax

Operator	Meaning	Example	Matches
.	Any single character except new line	c.t	cat, cot, cut
*	0 or more repetitions	lo*I	ll, lol, loool
+	1 or more repetitions.		
+	Equivalent to {1,}	go+gle	gogle, google, google
?	Optional (0 or 1)	colou?r	color, colour
[]	Character class	[aeiou.]	a, e, o (Any single not capital vowel or dot)
[n-m]	Character class with a range	[a-z]	q, z, c, g, j (Any single not capital letter)
[^] and [^n-m]	Negated character class	[^abc]+ or [^a-c]+	snow, low (Anything that does not have a, b or c)
^	Matches the beginning of a string	^He	"Helicopter", "Hello"
\$	Matches the end of a string	end\$	"the end", "a legend"
()	Grouping / capturing	(ab)+	ab, abab
x{n}	Matches exactly n	a{2}	aa

	occurrences of x		
x{n,}	Matches at least n occurrences of x	a{2, }	aa, aaa, aaaa
x{n,m}	Range repetition of x	\d{2,4}	11, 321, 7890 (2 to 4 digits)
(x y)	Matches either "x" or "y". Equivalent to OR	(green red).[a-z]+	"redbull", "green apple", "red apple"
x(?=y)	Matches x only if x is followed by y	Hello(?=World)	"Hello" (Full text is "HelloWorld", "World" is not included in the match result)
x(?!y)	Matches x only if x is not followed by y	\d+(?!.)	"0", "14" (Full text is "03.14", "3" is excluded because there is a dot after it)
(?<=y)x	Matches x only if x is preceded by y	(?<=Hello)World	"World" (Full text is "HelloWorld", "Hello" is not included in the match result)
(?<!y)x	Matches x only if x is not preceded by y	(?<!-)\d+	"73", "15" (Full text is "-273.15", "2" is excluded because there is a minus in front of it)
\b	Word boundary	\bcat\b	"cat" (Doesn't work for "catfish", "tomcat" or "concatenate")
\B	Not a word boundary	\Bcat\B	"concatenate" (Doesn't work for "catfish", "tomcat" or "cat")

If we want "-" (hyphen) included in a class we have to put it first or last in the brackets - [-abc], [^abc-].

If ? is used immediately after *, +, ?, or {}, it makes the quantifier non-greedy (matching the minimum number of times), as opposed to the default, which is greedy (matching the maximum number of times)

Operator	Equivalent
\d	[0-9]
\D	[^0-9]
\w	[A-Za-z0-9_]
\W	[^A-Za-z0-9_]
\s	Matches a single white space character, including space, tab, form feed, line feed, and other Unicode spaces
\S	Matches a single character other than white space
\t	Matches a horizontal tab
\v	Matches a vertical tab
\n	Matches a linefeed
\r	Matches a carriage return

3. RegEx functions

RegEx functions in SQL were created to make pattern based text searching and manipulation inside the database easier without needing to extract the data into a separate script / program

There are three main functions used in SQL to work with regular expressions:

REGEXP_LIKE(column_name, 'pattern')

The **REGEXP_LIKE** function in SQL is used to determine whether a given string matches a specific **regular expression pattern**. It evaluates the string against the regex and returns **TRUE** if the string matches the pattern and **FALSE** otherwise.

REGEXP_REPLACE (string, 'pattern', 'replacement')

The **REGEXP_REPLACE** function in SQL is used to search for a pattern within a string and replace all occurrences of that pattern with a specified replacement string. This function is particularly useful for **cleaning** and **transforming data** by removing unwanted characters or formatting strings.

REGEXP_SUBSTR (string, 'pattern', start_position, occurrence, match_parameter)

The **REGEXP_SUBSTR** function in SQL is used to extract a substring from a string that matches a specified regular expression pattern. It is particularly useful for isolating portions of text within a larger string, such as extracting domain names from email addresses or pulling specific elements from formatted data.

4. Differences between different types of SQL

Similarly to SQL RegEx has different "flavors" (various syntaxes and features) such as POSIX, Perl-compatible (PCRE) and others. While the core concepts are the same, they differ in how special characters are handled, the features they support (like lookaheads and named capture groups), and the specific metacharacters they use.

While not all SQL databases support regex directly, many popular ones do, often using extensions or specific functions. The syntax for regular expressions in SQL can vary slightly between database systems. MySQL implements regular expression support using International Components for Unicode (ICU). PostgreSQL uses POSIX-style

regular expressions. Oracle supports the use of Perl Compatible Regular Expressions (PCRE) syntax. This means that a regex pattern that works in one database might not work in another without modification. For example in PostgreSQL using \y for word boundaries will need to be rewritten for MySQL or Oracle.

MySQL: Uses the REGEXP (or RLIKE) operator for pattern matching.

PostgreSQL: Uses specific operators for case-sensitive (~) and case-insensitive (~*) matching.

Oracle: Provides a set of functions like REGEXP_LIKE, REGEXP_INSTR, REGEXP_REPLACE, and REGEXP_SUBSTR.

SQL Server: Does not support regex natively, but it can be simulated.

5. Practical example / quiz

RegEx functions - practical examples

let's say we've created a table called users in which we have emails stored for each user, but we have an imposter:

	id	email
1		name.surname@gmail.com
2		ivanov123@yahoo.com
3		fake_email@@test

REGEXP_LIKE

Now imagine that we want to extract all real emails from our table, how would we do that and what would our regular expression look like?:

Answer:

```
SELECT email
FROM users
WHERE REGEXP_LIKE(email, '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}+$');
```

REGEXP_REPLACE

What if we want to anonymize the data of our users so the domain of their email is not known by replacing it with an imaginary one like @example.com?

Answer:

```
SELECT REGEXP_REPLACE(email, '@.*', '@example.com') AS masked_email
FROM users;
```

REGEXP_SUBSTR

What if we want to extract just the domain names of our users' email addresses

Answer:

```
SELECT REGEXP_SUBSTR(email, '@[A-Za-z0-9.-]+') AS domain
FROM users;
```

Quiz timeeeeeeee :)