

# Introduction to Regular Expressions in SQL Server

**Erland Sommarskog**

Data Platform MVP

# Erland Sommarskog

SQL Server MVP since 2001

Independent consultant based in Stockholm



[esquel@sommarskog.se](mailto:esquel@sommarskog.se)



<https://www.sommarskog.se>

Slides and scripts: <https://www.sommarskog.se/present>



# Agenda

- Introduction and overview.
- A number of Lessons – done in SSMS and key points repeated on slides.

# Regular Expressions, What? Why?

- What: Advanced search patterns for string data.
- Why: Advanced string manipulation.
- Example: replace multiple spaces with a single:  

```
SELECT regexp_replace(  
  'A    text with    extra    space', ' +', ' ')
```
- Returns `A text with extra space`.
- Doing this in SQL 2022 is very difficult!

# Regexps – Caveat about Implementations

- About every environment that has regular expressions has its own implementation.
- =>Just like SQL, there are differences between environments.
- SQL Server's implementation is based on the [RE2](#) library from Google.
- RE2 is fairly basic – still powerful enough.
- There are some irritating deviations.

# Overview of the RegExp Functions

- REGEXP\_LIKE – Does *string* match *regexp*?
- REGEXP\_REPLACE – Like the REPLACE function, but the search string can be a regular expression.
- REGEXP\_COUNT – How many matches of *regexp* are there in *string*?
- REGEXP\_SUBSTR – Returns one match of *regexp* in *string* – by default the first, but you can get others.
- REGEXP\_INSTR – Returns starting or ending position of *regexp* in *string*.

# Two Table-valued Functions

- REGEXP\_MATCHES – Returns a table with all matches of *regexp* in *string*.
  - Useful as a debug aid.
- REGEXP\_SPLIT\_TO\_TABLE – Returns a table with the fragments of *string* outside matches of *regexp*.
  - This is STRING\_SPLIT on steroids!

# Notes on Compatibility Level

- REGEXP\_LIKE and the table-valued functions require compatibility level **170**.
- The other functions – REGEXP\_REPLACE, REGEXP\_COUNT, REGEXP\_SUBSTR and REGEXP\_INSTR – work on any compatibility level.
- On lower compatibility levels, REGEXP\_COUNT can be a decent substitute for REGEXP\_LIKE.



# Data-Type Support

Regex functions support string types in SQL Server with these restrictions:

- The old types **text** and **ntext** are not supported.
- REGEXP\_LIKE, REGEXP\_COUNT and REGEXP\_INSTR support **(n)varchar(MAX)** for input *up to 2 MB*.
- REGEXP\_SUBSTR also accepts MAX input – but documentation does not say so. Use at own risk.
- Other functions do “currently” not support MAX input.

# Lesson One – Comparison with LIKE

[regexdemos.sql](#)

- REGEXP\_LIKE is a boolean function.
  - Used like LIKE, but it has a function syntax.
  - Cannot be used in a SELECT list directly, wrap in IIF or similar.
- With LIKE, the search pattern must describe the full search string.
- A regular expression can match anywhere in the search string.

# Lesson One – Case Sensitivity

- LIKE respects and understands collations.
- Regular expressions *do not* and are by default case-sensitive.
- Set the **flags** parameter to **i** for case-insensitive.
- All regexp functions accept **flags** – position varies.

# Lesson One - Metacharacters

- In regexps, a number of punctuation characters serve as *metacharacters* with special meaning.

First few metacharacters:

- ^ – Matches at the beginning of string.
- \$ – Matches at the end of string.
- . (dot) – Matches any character exactly once.
  - Exception: newline.

# Lesson one – Metacharacter backslash

Backslash (\) is an escape character.

- Before an ASCII punctuation character it means “take next character at face value”.
- Best practice: To match ASCII punctuation chars, always escape with backslash to avoid surprises.
- Before non-ASCII character, you get an error.
- Backslash + ASCII alphanumeric may have a special meaning, as we will look at later.
  - If there is no special meaning, you get an explicit error.

# Lesson Two – Brackets

[regexdemos.sql](#)

- Both with LIKE and regular expressions you can match a group of characters with help of brackets.
- **[abc]** – Matches any of "a", "b" and "c".
- **[0-9]** – Matches all characters in the range.
  - With regexps, the range is defined from character code points.
- **[^abc0-9]** – The ^ negates the set; Matches anything *but* "a", "b", "c" and digits.

# Lesson Three – Quantifiers

[regexdemos.sql](#)

- Quantifiers is the core of why regular expressions are so powerful!
- A quantifier specifies how many times the previous regular expression needs to appear for a match.

The most common:

- \* (star) – 0 or more times.
- + (plus) – 1 or more times.
- ? (question mark) – 0 or 1 time.

# Lesson Three – Quantifiers and Parentheses

The general ones:

- $\{m\}$  – Match exactly  $m$  times.
- $\{m,n\}$  – Match at least  $m$  and at most  $n$  times.
- $\{m,\}$  – Match  $m$  or more times.

Use parentheses  $()$  to change precedence.

- $(st)^+$  – match **st** once or more.
- $st^+$  – match **s** exactly once and **t** once or more.



# Lesson Four – Alternates

[regexdemos.sql](#)

- The metacharacter | (pipe) separates alternate regular expressions that qualify for a match.
- E.g. **(abc|jkl|xyz)** = any of "abc", "jkl" and "xyz" matches.
- Because the pipe has lowest precedence of the regexp operators, always enclose a list of alternates in parentheses.

# Lesson Five – REGEXP\_REPLACE

[regexdemos.sql](#)

- In the replacement string, use `\&` to specify that the entire match should be inserted into the return value.
  - Useful if you want to wrap a match with something.
- Parentheses in the regexp define *capture groups*, which you can refer to in the replacement string as `\1`, `\2` etc. Very powerful!
- To insert backslash in replacement string, use `\\`.

# Lesson Five – REGEXP\_REPLACE, Parameters

- Fourth parameter: **start**. Start replacing from this position (one-based). Default: 1.
- Fifth parameter: **occurrence**. Replace only this occurrence (one-based). Default: 0 (replace all).
- Sixth parameter: **flags**. **i** to force case-insensitive.
  - To be able to specify **flags**, you must also specify **start** and **occurrence**. Set them to 1 and 0.
  - If you set them to NULL – you get NULL back!

# Lesson Five - Greediness

- Quantifiers are by default “greedy”.
  - They match as long as they can, and only then the RE engine starts backtracking to find matches for the for the rest of the regexp.
- Override this behaviour by putting a **?** after the quantifier.
  - Now it will stop when the rest of the regexp matches.
- Tip: Write your regexp without any extra **?**, but try adding **?** if a quantifier appears to be greedy.

# Lesson Six – Escape Sequences

[regexdemos.sql](#)

- Shortcuts to specify control characters in regexps:
  - **\t** = Tab.
  - **\r** = Carriage return.
  - **\n** = Line feed (a.k.a newline).
  - **\f** = Form feed.
- To specify any character, use **\x{nnnnnnnn}**, where *nnnnnnnn* is the Unicode code point in hex.
  - Leading zeroes can be left out.
  - Braces not needed for values  $\leq$  FF.

# Lesson Six – Character Classes

- **\s** – Matches white space: Space, tab, CR, LF and FF – but not hard space.
  - To also match hard space, use **[\s\xA0]**.
- **\S** – Matches everything that **\s** does not match.
- **\d** – Matches digits 0-9. I.e, the same as **[0-9]**.
  - Does not match “fancy” digits.
- **\D** – Matches everything that **\d** does not match.

# Lesson Six – More Character Classes

- **\pL** matches characters classified as letters in Unicode.
  - **\p{Lu}** matches uppercase letters.
  - **\p{Li}** matches lowercase letters.
- **\PL** matches everything that is not a letter. Useful when you want to match a whole word in text.
  - Example: **(^|\PL)word(\PL|\$)**
- More generally, **\pX** matches a Unicode category and **\p{Xx}** matches a subcategory – 30+ of them.
  - E.g. **\p{Nd}** matches digits, including “fancy” digits.
  - Link to reference on the last slide.

# Lesson Six – The Ugly Classes

- **\w** – Matches word characters, i.e. digits, letters and underscore, but is ASCII only. **DO NOT USE!**
- **\W** – The inverse of **\w**. **DO NOT USE!**
- **[[:class:]]** – Posix classes. ASCII only. **DO NOT USE!**
- These restrictions are specific to SQL Server (and the Google library). These operators work correctly on other platforms.



# Lesson Seven – Handling Line Breaks

[regexdemos.sql](#)

- The dot (.) matches all characters *but line feed*.
- You can override this by setting **flags** to **s**.
  - Or **is**, if you also want force case-insensitive.
- To match a line break in a multi-line string, use **\r?\n**, to handle both Windows and Unix input.

# The Bottom Line

- With regular expressions you can perform advanced string operations that previously were impossible or very cumbersome to write in T-SQL.
- Complex regular expressions can certainly be deterrently hard to read. You will get used to it. :-)
- Be warned: You will still hit limits. Not all parsing is fit for regular expressions.
- It still a giant leap forwards.

# Your feedback is important to us



**Evaluate this session at:**

[www.PASSDataCommunitySummit.com/evaluation](http://www.PASSDataCommunitySummit.com/evaluation)

# THANK YOU!

Erland Sommarskog



esquel@sommarskog.se

## References:

- Regular Expressions Overview:  
<https://learn.microsoft.com/en-us/sql/relational-databases/regular-expressions/overview>.
- Syntax Google RE2:  
<https://github.com/google/re2/wiki/Syntax>.
- Slides and scripts:  
<https://www.sommarskog.se/present>.