ISO/IEC JTC1 SC32 WG3 W28-015
2024-11-01

Title:	W28-015: Pattern matching versus collations
Date:	2024-11-01
Author:	Peter Eisentraut
Status:	discussion

Abstract

Clarifications, open questions, and possible problems in how collations are applied to pattern matching predicates.

1 Discussion

SQL/Foundation has three pattern matching predicates:

- LIKE (subclause 8.5 <like predicate>)
- SIMILAR (subclause 8.6 <similar predicate>)
- LIKE_REGEX (subclause 8.7 <regex like predicate>)

These apparently take collations into account in different ways. In this paper, I want to seek clarification of these and raise possible problems.

1.1 LIKE

LIKE works by partitioning the pattern into wildcards and non-wildcard strings and then checking if a partioning of the string to be matched exists that can be matched up to the partitioned pattern. So

```
'foobar' LIKE 'foo%'
```

is true because 'foo' = 'foo' and 'bar' matches '%'. Here, the 'foo' = 'foo' comparison uses the applicable collation.

1.2 SIMILAR

The specification of SIMILAR works differently. Here, we define the set of all possible strings described by the regular expression and then check if the string to be matched is in that set. So

```
'foobar' SIMILAR TO 'foo%'
```

is true because the regular expression 'foo%' describes the set of strings {'foo', 'fooa', 'foob', ..., 'foobar', ..., 'fooxyz', ...}, and it turns out there is a string in that set that is equal to 'foobar' under the applicable collation.

1.3 Different behavior between LIKE and SIMILAR

These two ways of defining matching might look equivalent, but they are not. Consider a collation that considers 'ss' to be equal to 'ß'. (This is not usually the default for German language collations, but it would not be unreasonable to define such a collation for search operations.) With such a setup,

```
'ß' LIKE 's_'
```

is false. The partitioning of the pattern is ('s', '_'), and there is no way to make a partitioning of 'ß' to match that. But

```
'ß' SIMILAR TO 's_'
```

is true, because the set of strings described by 's_' includes 'ss', which is equal to 'ß' here.

Similar examples can be constructed wherever strings of different lengths are considered equal in a collation.

Note that in the general case, implementing this behavior for SIMILAR would be quite expensive, because the system would have to check all possible expansions of the involved wildcards to see if there is a possible match. (This is not the case for LIKE.)

Both subclause 8.5 and 8.6 contain language that it is implementation-defined which collations can be used for LIKE and SIMILAR, respectively, to allow implementations to avoid some of these complications.

I'm showing these differences merely to point out that there is no single obvious way to handle collations in pattern matching.

1.4 LIKE_REGEX

LIKE_REGEX by contrast makes no specifications about collations at all. There is no language that a collation is to be resolved or which collations are permitted, and no collation is applied during the matching. (Also, the details of the matching are handled by the XQuery specification, which has no concept like SQL collations.)

Maybe this is intended, but perhaps it could be made more explicit, for the benefit of both implementers and users.

Also, existing implementations appear to take different approaches for their analogous features. For example, in Oracle Database, REGEXP_LIKE() matches case-insensitively if the applicable collation is BINARY_CI. This makes some sense because then you get consistent behavior between LIKE and REGEXP_LIKE. But it's unclear how to generalize that to arbitrary collations, considering the difficulties mentioned at the SIMILAR example.

1.5 Unicode guidance

It appears that the trend in Unicode is that regular expressions should not have language-specific behavior. Notably, the "tailoring" support in UTS #18 has been removed in recent versions. (Compare for example the old version <https://www.unicode.org/reports/tr18/tr18-19.html#RL3.2>

versus the current version <https://www.unicode.org/reports/tr18/#Tailored_Support>). Also note that XQuery also refers to UTF #18, so this is technically in scope for SQL.

1.6 Possible outcomes

I wonder what other WG3 participants think the right behavior should be. And how we can clarify that. Some possibilities:

- Add a note in subclause 8.7 and/or in other related subclauses that the applicable collation is not considered. This is formally the same as the current behavior.
- Add specification language that and to what extent the applicable collation is used is implementation-defined. This would allow implementers to support things like case-insensitive matching based on a case-insensitive collation.

--- end of the paper ---