L2/00-260

Author: Gary Roberts
Date: 8th of August, 2000

Unicode Identifiers: an SQL Example

While any standards body is free to define their standard as they wish, it is clear that the Unicode consortium can have a positive effect on uniformity and quality on other standards. The purpose of this paper is to stimulate discussion towards a set of recommendations concerning identifiers. I use the SQL-99 standard as an example source for such issues, not because it is either good or bad, but because it is a standard that I have recently examined, and it raises several interesting issues. I have divided these into two sets of issues. The first is the utility of Unicode properties in SQL identifiers. The second is the usage of normalization (case normalization in the SQL standard) in SQL identifiers.

The following extract from the SQL-99 standard concerns the use and equivalence of identifiers. There are two types of identifiers defined in the standard: the <regular identifier> and the <delimited identifier>. A <regular identifier> is constructed from a limited repertoire of characters known as the <identifier body>. A <delimited identifier> is essentially anything inside double quotes.

The following BNF can be used as reference for the rest of the document.

```
<identifier body> ::=
       <identifier start>
       [ { <underscore> | <identifier part> }... ]

<identifier start> ::=
       <initial alphabetic character>
       | <ideographic character>

<identifier part> ::=
       <alphabetic character>
       | <ideographic character>
       | <decimal digit character>
       | <identifier combining character>
       | <underscore>
       | <alternate underscore>
       | <extender character>
       | <identifier ignorable character>
       | <connector character>

<delimited identifier> ::=
       <double quote> <delimited identifier body> <double quote>

<delimited identifier body> ::= <delimited identifier part>...

<delimited identifier part> ::=
       <nondoublequote character>
       | <doublequote symbol>
```

This part of the document raises issues on the utility of properties in the definition of identifiers.

1) An <alphabetic character> is any character with the Unicode alphabetic property.

   NOTE 35 – Characters with the ‘‘alphabetic’’ property include characters that are called ‘‘letters’’ and characters that are called ‘‘syllables’’.

Pretty clear.

2) An <initial alphabetic character> is an <alphabetic character> that does not have the Unicode ‘‘combining’’ property.

No Problem

3) An <ideographic character> is a character with the Unicode ‘‘ideographic’’ property.

No Problem

4) A <decimal digit character> is a character with the Unicode ‘‘decimaldigit’’ property.

No Problem

5) An <identifier combining character> is a character with the Unicode ‘‘combining’’ property, except for U+06DD, U+06DE, U+20DD, U+20DE, U+20DF, and U+20E0.

NOTE 36 – U+06DD is ‘‘Arabic End of Ayah’’, U+06DE is ‘‘Arabic Start of Rub El Hizb’’, U+20DD is ‘‘Combining Enclosing Circle’’; U+20DE is ‘‘Combining Enclosing Square’’, U+20DF is ‘‘Combining Enclosing Diamond’’, and U+20E0 is ‘‘Combining Enclosing Circle Backslash’’.

This is unfortunate.

6) An <extender character> is any of U+00B7, U+02D0, U+02D1, U+0640, U+0E46, U+0EC6, U+3005, U+3031 through U+3035 inclusive, U+309B through U+309E inclusive, U+30FC through U+30FE inclusive, U+FF70, U+FF9E, and U+FF9F.

NOTE 37 – U+00B7 is ‘‘Middle Dot’’, U+02D0 is ‘‘Modifier Letter Triangular Colon’’, U+20D1 is ‘‘Modifier Letter Half Triangular Colon’’, U+0640 is ‘‘Arabic Tatweel’’, U+0E46 is ‘‘Thai Character Maiyamok’’, U+0EC6 is ‘‘Lao Ko La’’, U+3005 is ‘‘Ideographic Iteration Mark’’, U+3031 through U+3035 inclusive are variations of Vertical Kana Repeat Marks, U+309B through U+309E inclusive are variations of Combining Katakana-Hiragana Sound Marks and Hiragana Iteration Marks, U+30FC through U+30FE inclusive are variations of Katakana-Hiragana Prolonged Sound Mark and Katakana Iteration Marks, U+FF70 is ‘‘Halfwidtyh Katakana-Hiragana Prolonged Sound Mark’’, U+FF9E is ‘‘Halfwidth Katakana Voiced Sound Mark’’, and U+FF9F is ‘‘Halfwidth Katakana Semi-voiced Sound Mark’’.

Should we have an extender property?

7) An <identifier ignorable character> is any of U+200C through U+200F inclusive, U+202A through U+202E inclusive, U+206A through U+206F inclusive, and U+FEFF.

NOTE 38 – U+200C is ‘‘Zero Width Non-Joiner’’, U+200D is ‘‘Zero Width Joiner’’, U+200E is ‘‘Left-To-Right Mark’’, U+200F is ‘‘Right-To-Left Mark’’, U+202A is ‘‘Left-To-Right Embedding’’, U+202B is ‘‘Right-To-Left Embedding’’, U+202C is ‘‘Pop Directional Formatting’’, U+202D is ‘‘Left-To- Right Override’’, U+202E is ‘‘Right-To-Left Override’’, U+206A is ‘‘Inhibit Symmetric Swapping’’, U+206B is ‘‘Activate Symmetric Swapping’’, U+206C is ‘‘Inhibit Arabic Form Shaping’’, U+206D is ‘‘Activate Arabic Form Shaping’’, U+206E is ‘‘National Digit Shapes’’, U+2006F is ‘‘Nominal Digit Shapes’’, and U+FEFF is ‘‘Zero-Width No-Break Space’’.

Again, no principle.

8) An <alternate underscore> is any of U+FE33, U+FE34, U+FE4D, U+FE4E, U+FE4F, and U+FF3F.

NOTE 39 – U+FE33 is ‘‘Presentation Form for Vertical Low Line’’, U+FE34 is ‘‘Presentation Form for Vertical Wavy Low Line’’, U+FE4D is ‘‘Dashed Low Line’’, U+FE4E is ‘‘Centreline Low Line’’, U+FE4F is ‘‘Wavy Low Line’’and U+FF3F is ‘‘Fullwidth Low Line’’.

One of Asmus’s folding properties?

9) A <connector character> is any of U+203F or U+2040.

NOTE 40 – U+203F is ‘‘Undertie’’ and U+2040 is ‘‘Character Tie’’.

Principle?

The second part of this document raises issues on how to correctly handle identifiers.

21) For every <identifier body> *IB* there is exactly one corresponding case-normal form *CN*F. *CNF* is an <identifier body> derived from *IB* as follows.

Let *n* be the number of characters in *I*B. For *i* ranging from 1 (one) to n, the i-th character *M i* of *IB* is translated into the corresponding character or characters of *CNF* as follows.

Case:

a) If *M i* is a lower case character or a title case character for which an equivalent upper case sequence *U* is defined by Unicode, then let *j* be the number of characters in U; the next *j* characters of *CNF* are U.

b) Otherwise, the next character of *CNF* is *M i* .

CNF converts all strings to upper case. It appears to be not fully integrated into the standard, but presumably, when the standard mentions conversion of lower-case to upper-case, the formal technique is conversion to CNF.

22) The case-normal form of the <identifier body> of a <regular identifier> is used for purposes such as and including determination of identifier equivalence, representation in the Definition and Information Schemas, and representation in diagnostics areas.

NOTE 44 – Any lower-case letters for which there are no upper-case equivalents are left in their lower-case form.

Item 22 is essentially equivalent to item 23. It essentially says that the normalized form is what is displayed to the user rather than the form that the user typed. In my opinion, this is misguided.

23) The <identifier body> of a <regular identifier> is equivalent to an <identifier body> in which every letter that is a lower-case letter is replaced by the equivalent upper-case letter or letters. This treatment includes determination of equivalence, representation in the Information and Definition Schemas, representation in the diagnostics area, and similar uses.

Item 23 is essentially equivalent to item 22.

24) The <identifier body> of a <regular identifier> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters), treated as the repetition of a <char-acter string literal> that specifies a <character set specification> of SQL_IDENTIFIER, shall not be equal, according to the comparison rules in Subclause 8.2, ‘‘<comparison predicate>’’, to any <reserved word> (with every letter that is a lower-case letter replaced by the correspond-ing upper-case letter or letters), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_IDENTIFIER.

NOTE 45 – Assurance that a <regular identifier> will not become a <reserved word> in a possible future revision of ISO/IEC 9075 can be obtained by including a <digit> or an <underscore> in the <regular identifier>, and by taking care to avoid identifiers beginning with CURRENT_, SESSION_, SYSTEM_, or TIMEZONE_, or ending in _LENGTH.

Regular identifiers are compared against all possible SQL keywords by converting to upper-case, and rejected if they match.

25) Two <regular identifier>s are equivalent if their <identifier body>s, considered as the rep-etition of a <character string literal> that specifies a <character set specification> of SQL_IDENTIFIER, compare equally according to the comparison rules in Subclause 8.2, ‘‘<compari-son predicate>’’.

The comparison rules in Subclause 8.2 do not quickly lead to a resolution of how comparison is performed. It is my understanding that a collation that is not sensitive to case is used to determine equality.

26) A <regular identifier> and a <delimited identifier> are equivalent if the <identifier body> of the <regular identifier> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters) and the <delimited identifier body> of the <delimited identifier> (with all occurrences of <quote> replaced by <quote symbol> and all occurrences of <doublequote symbol> replaced by <double quote>), considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_IDENTIFIER and an implementation-defined collation that is sensitive to case, compare equally according to the comparison rules in Subclause 8.2, ‘‘<comparison predicate>’’.

Regular identifiers are converted to upper-case, and compared to delimited identifiers with a case-sensitive comparison to check if they are equal (e.g., “ABC” = ABC = abc <> “abc”). This does not sit well with me.

27) Two <delimited identifier>s are equivalent if their <delimited identifier body>s, considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_IDENTIFIER and an implementation-defined collation that is sensitive to case, compare equally according to the comparison rules in Subclause 8.2, ‘‘<comparison predicate>’’.

Delimited identifiers are considered identical if a case-sensitive comparison determines they are equal.

28) For the purposes of identifying <key word>s, any <simple Latin lower case letter> contained in a candidate <key word> shall be effectively treated as the corresponding <simple Latin upper case letter>.

Key words are converter to upper case before comparison (e.g. MESSAGE and meßage). This seems a bit odd.