per054r1.doc
Published at 9:56 am on 20 April 2001

WG3: PER-054R1 L2/01-258

18 April 2001

Authoritative version: per054r1.pdf

ISO

International Organization for Standardization

British Standards Institution
IST/40
Data Management and Interchange
ISO/IEC JTC 1/SC 32
Data Management and Interchange
WG 3
Database Languages

Project: 32.03.05.02.00

Title: SQL support for the Universal Character Set

Author: J M Sykes (United Kingdom)

Source: UK Expert

Status: Comment resolution

Abstract: **This paper is a revision of the proposal previously posted informally under the file name jms01v7. Its intention is to resolve some of the problems relating to character sets in SQL that were discussed in earlier papers. It is still incomplete. The other paper previously posted, jms01v6, represented an approach which is now felt to be more complex than necessary; it has been abandoned, but individual features could be resurrected and reinstated if required.**

References:

[FrameworkCD] *(ISO Committee Draft) Database Language SQL - Part 1*: *Framework (SQL/Framework*), November 2000

[FoundCD] *(ISO Committee Draft) Database Language SQL - Part 2: Foundation (SQL/Foundation*), November 2000

[CLI-CD] *(ISO Committee Draft) Database Language SQL - Part 3: Call-Level Interface (SQL/CLI*), November 2000

[PSM-CD] *(ISO Committee Draft) Database Language SQL - Part 4: Persistent Stored Modules (SQL/PSM*), November 2000

[MED-CD] *(ISO Committee Draft) Database Language SQL - Part 8: Management of External Data (SQL/MED*), November 2000

[OLB-CD] *(ISO Committee Draft) Database Language SQL - Part 9: Object Language Bindings (SQL/OLB*), November 2000

[SchemataCD] *(ISO Committee Draft) Database Language SQL - Part 10: Schemata (SQL/Schemata)*, November 2000

[10646] ISO/IEC 10646-1:2000, Information technology - Universal Multi-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane, May 2000.

[14651] ISO/IEC 14651:2001 – International string ordering and comparison – Method for comparing character strings and description of the common template tailorable ordering (approved for publication).

[Unicode3] The Unicode Consortium, The Unicode Standard, Version 3.0, Reading, MA, Addison-Wesley Developers Press, 2000. ISBN 0-201-61633-5.

[Unicode3.1] The Unicode Consortium, *The Unicode Standard, Version 3.1,* [Update to The Unicode Standard, Version 3.0 as described on http://www.unicode.org/unicode/standard/versions/]

[UTR#15] Unicode Standard Annex #15, Unicode Normalization Forms, Version 3.1. Cupertino, CA. June, 1999. The Unicode Consortium. http://www.unicode.org/unicode/reports/

[UTR#19] Unicode Standard Annex #19, UTF-32, Version 3.1, Cupertino, CA. June, 1999. The Unicode Consortium. http://www.unicode.org/unicode/reports/

[UTR#10] Unicode Technical Standard #10: Unicode Collation Algorithm, Version 3.1. Cupertino, CA. June, 1999. The Unicode Consortium. http://www.unicode.org/unicode/reports/

[UTR#18] Unicode Technical Report #18, Unicode Regular Expression Guidelines, Version Cupertino, CA. June, 1999. The Unicode Consortium. http://www.unicode.org/unicode/reports/

[HEL-052] *Possible problems with characters, the story continues*, Sykes, September 2000 (WG3: HEL-052)

[PER-095] *Changes to SchemataCD for character sets*, Sykes, April 2001 (WG3: PER-095)

1 Introduction

The purpose of this paper is specify the proposals outlined in [HEL-052].

Section 2 of this paper recapitulates issues considered in earlier papers; Section 3 discusses issues which have arisen subsequently.

Section 4 specifies the proposals for [FoundCD]. Corresponding changes to [SchemataCD] are proposed in [PER-095].

2 Issues previously considered - a recap

2.1 <character set name>s

There is a problem in that there is no distinction in SQL between *repertoire* and *coded character set*. The SQL term *character set* appears to mean mostly the former (though the term *repertoire* is often used explicitly), but sometimes what UTR #17 calls a *character encoding form* (especially in clause 4.2.4 "Named character sets"), and which one is intended is not always clear from the context.

To minimise perturbation, we have taken the approach that an SQL *character set* is a *character encoding form*, i.e. a repertoire, encoding (into *code points*) and transformation format (into *code units*). This fits with "Named character sets", and makes an SQL character set virtually the same as an IANA *charset*, but leaves the awkwardness described above: several character sets can encode the same repertoire. Indeed, *all* the character sets mentioned in Clause 4.2.4 have repertoires that are subsets of UCS.

This interpretation also has the advantage that a character string type tells a host language everything it needs to know for purposes of data interchange.

2.2 Normalization

We have concluded that normalization form should *not* be regarded as a characteristic of a declared type, partly because a string value may conform to more than one normalization form, but also for the sake of simplicity.

The SQL-implementation is permitted to assume character data is in normalization form C and the user is provided with a predicate to test for it (in a constraint, for example) and a function to normalize. All the SQL-implementation is required to do is *not* to return an unnormalized concatenation result from normalized arguments.

2.3 Collations

We propose no change to <collate clause> or to the places where it may occur.

We note the opportunity to associate a specific, possibly culture-dependent collation with an SQL-session, but do not pursue it here.

2.4 Counting characters

For data type definition and for operations such as POSITION, SUBSTRING, the user is permitted to specify whether the units are to be bytes, code units or code points, with code points being the default. The possibility of allowing graphemes (sequences of one or more code points) was considered, but deferred until an established definition of grapheme is available.

3 Issues not previously considered

3.1 A "UCS support" feature for SQL

We propose a feature F*nnn* (F471 is suggested) "UCS support", which is intended to embody all the rules required for processing UCS data. It will clearly be dependent on F461, "Named character sets", but will *not* be dependent of either of the existing features F451 "Character set definition" and F691, "Collation and translation", except to the extent that it *will* use <collate clause>, which will be compatibly enhanced for this feature.

3.2 Terminology

In specifying UCS support, we have introduced some terms defined in existing normative references; as far as possible, we have selected the terms that seem to:

a) be most easily distinguished from each other, and

b) be in wide use already.

3.3 The correspondence between character sets and collations in SQL

If, as explained in section 2.1 above, we proceed on the basis that UTF8, UTF16 and UTF32 are predefined character set names, then it follows that any collation that can be applied to one can *ipso facto* be applied to any of the three, because they all encode the same repertoire. (Moreover, we observe that [ISO 14651] is drafted so as to be encoding independent, so that any collation that conforms to it can, after any required transcoding, be applied to any character set. However, we choose to ignore this for the moment.)

This does not conform to the present model of SQL, which expects a collation to apply to one and only one character set. This is most easily illustrated by the following quote from [FoundCD], Clause 4.2.1 "Character strings and collating sequences":

> For every character set, there is at least one collation. A collation is described by a collation descriptor. A collation descriptor includes:
>
> - The name of the collation.
> - The name of **the** character set on which the collation operates.

(The breadth of applicability of a collation also calls into question the appropriateness of associating every character set with a default collation - we choose not to pursue this issue.)

The problem with the existing model is more embarrassing in [SchemataCD], because DATA_TYPE_DESCRIPTOR base table relies on <collation name> to imply <character set name> (see, for example, the join

```
DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
   LEFT JOIN
   DEFINITION_SCHEMA.COLLATIONS AS C1
```

in COLUMNS view).

The change to the specification of collation descriptor is included in this paper. Proposals to change the definitions of Schemata base tables and views are in [PER-095].

4 Proposal for [FoundCD]

Proposed deletions are ~~struckout in red~~ while new or replacement text is in **bold blue** in the hope of helping reviewers.

4.1 Clause 2 Normative references

Note to Editor: These changes are only what appear to be correct at March 2001. They may well have been superseded by the time publication is approved.

Modify

ISO/IEC 10646-1:~~1993~~ **2000**, *Information technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane*.

Add:

ISO/IEC 14651:2000 – International string ordering and comparison – Method for comparing character strings and description of the common template tailorable ordering

Modify:

The Unicode Consortium, *The Unicode Standard, Version ~~2~~3.0*, 2000. ISBN 0-201-~~48345-9~~**61633-5**.

Add:

The Unicode Consortium, *The Unicode Standard, Version 3.1*, [Update to The Unicode Standard, Version 3.0 as described on http://www.unicode.org/unicode/standard/versions/]

Unicode Standard Annex #15, Unicode Normalization Forms

Unicode Standard Annex #19, UTF-32

***Unicode Technical Standard #10: Unicode Collation Algorithm, Version 3.1*.**
Cupertino, CA. June, 1999. The Unicode Consortium. http://www.unicode.org/unicode/reports/

4.2 Clause 3.1 Definitions

4.2.1 Clause 3.1.1 Definitions taken from ISO/IEC 10646

Add:

repertoire

Note to proposal reader: the definition is **repertoire:** A specified set of characters that are represented in a coded character set.

After the existing entry" character", add the following Note:

NOTE nnn - This is identical to the Unicode definition of *abstract character*. In ISO/IEC 9075, when the relevant character set is UTF8, UTF16 or UTF32, a character can be thought of as *that which is represented by one code point*.

Note to proposal reader: Here and elsewhere we have used "UTF8, UTF16 or UTF32" rather than, e.g. "a UCS character set" (which seems rather clumsy). Comments would be welcome.

4.2.2 Clause 3.1.n Definitions taken from ISO/IEC 14651

Insert a subclause with this title, and add:

collation

Note to proposal reader: the definition is **collation** equivalent to the term "ordering" The definition of "ordering" is: **ordering** a process by which two strings are determined to be in exactly one of the relationships of **less than**, **greater than**, or **equal** to one another

4.2.3 Clause 3.1.2 Definitions taken from Unicode

Add:

code point

Note to proposal reader: the definition is ***Unicode Scalar Value***. A number N from 0 to $10FFFF_{16}$ defined by application of the algorithm in Definition D28. (See Section 3.7, Surrogates.) Also known as a ***code point***. (This is taken from the Unicode Glossary, and is somewhat shorter than the full Definition D28.)

code unit

Note to proposal reader: the definition is ***Code Unit***. Synonym for *code value.* ***Code Value***. The minimal bit combination that can represent a unit of encoded text for processing or interchange. (See Definition D5)

noncharacter

Note to proposal reader: the following definition is in Unicode 3.1 ***Noncharacter****:* a code point that is permanently reserved for internal use, and that should never be interchanged. In Unicode 3.1, these consist of the values U+*n*FFFE and U+*n*FFFF (where *n* is from 0 to 10_{16}) and the values U+FDD0..U+FDEF.

4.2.4 Clause 3.1.5 Definitions provided in Part 2

Delete the following definitions:

~~*collation*~~ (definition is now taken from [14651])

~~*repertoire*~~ (definition is now taken from [10646])

~~*character repertoire*~~

4.3 Clause 4.2 Character strings

Modify paragraph 6 as follows:

Character sets fall into three categories: those defined by national or international standards, those defined by implementations, and those defined by applications.

The character sets defined by ISO/IEC 10646 and The Unicode Standard are known as Universal Character Sets (UCS) and their treatment is described in Clause 4.2.n "Universal Character Sets".

Every character set contains the <space> character (equivalent to U+0020). An application defines a character set by assigning a new name to a character set from one of the first two categories. They can be defined to "reside" in any schema chosen by the application. Character sets defined by standards or by implementations reside in the Information Schema (named INFORMATION_SCHEMA) in each catalog, as do collations defined by standards and collations, translations, and form-of-use conversions defined by implementations.

4.3.1 Clause 4.2.1 Character strings and collating sequences

Replace the second paragraph, "All character strings of a given character repertoire are comparable." with the following:

Two character strings are comparable if application of Subclause 4.2.3 "Rules determining collating sequence usage" produces a "Collating Sequence Used For The Comparison" other than "Not permitted: invalid syntax".

Modify item 2 of the last dash-list as follows:

- **A non-empty list of t**~~T~~he name**s** of the character set**s** to which the collation can be applied.

4.3.2 Clause 4.2.4 Named character sets

Delete item 4 in the dash-list, beginning:

~~UCS2 specifies the name of a character repertoire ...~~

Modify items 5 and 6 in the dash-list as follows:

- UTF8 ~~specifies the name of~~ **is** a character **set whose** repertoire ~~that consists of~~ **includes** every character ~~represented~~ **specified** by The Unicode Standard Version ~~2.0~~**3.1** and by ISO/IEC 10646 ~~UTF-16, where each character is~~ encoded ~~using the UTF-8 encoding~~ **as specified in ISO/IEC 10646-1, Annex D (normative), "UCS Transformation Format 8 (UTF-8)"**, ~~occupying either 2 or 4 octets~~ in which each character is encoded as from one to four code units, each of which is one octet.
- UTF16 ~~and ISO10646 specify the name of~~ **is** a character **set whose** repertoire ~~that consists of~~ **includes** every character ~~represented~~**specified** by The Unicode Standard Version ~~2.0~~**3.1** and by ISO/IEC 10646 ~~UTF-16, where each character is~~ encoded ~~using the UTF-16 encoding~~, **as specified in ISO/IEC 10646-1, Annex C (normative), "Transformation format for 16 planes of Group 00 (UTF-16)"**, ~~occupying either 2 or 4 octets~~ in which each character is encoded as one or two code units, each of which is two octets.

Add a new dash-list item:

- **UTF32 is a character set whose repertoire includes every character specified by The Unicode Standard Version 3.1 and by ISO/IEC 10646, encoded as specified in Unicode Standard Annex #19, "UTF-32", in which each character is encoded as one code unit of four octets.**

Take steps to register names.

Since SQL names cannot be hyphenated, we propose that the names UTF8 and UTF16 (already defined in [FoundCD]), together with UTF32, be registered with the appropriate authority, as aliases where appropriate of character sets already registered, and that the character sets so named be referred to collectively as *UCS character sets*.

4.3.3 Clause 4.2.5 Universal character sets (UCS)

Insert a new subclause with this title:

A UCS string is a character string whose character set is UTF8, UTF16 or UTF32. Any two UCS strings are comparable.

All UCS strings are assumed to be normalized in NFC, and an SQL-implementation may assume this to be the case. With the exception of <normalize function> and <normalized predicate>, the result of any operation on an unnormalized UCS string is implementation-defined.

Conversion of UCS strings from one character set to another is automatic.

Table n, "Character set of the result of a dyadic operation", shows how the character set is determined for the result of any dyadic operation. For comparison, both operands are converted as necessary to the character set of the result.

Table n- Character set of the result of a dyadic operation

Operand 1	Operand 2		
	UTF8	UTF16	UTF32
UTF8	UTF8	UTF16	UTF32
UTF16	UTF16	UTF16	UTF32
UTF32	UTF32	UTF32	UTF32

Detection of a noncharacter in a UCS-string causes an exception. The detection of an unassigned code point does not.

4.4 Clause 5.2 <token> and <separator>

Add the following to the definition of <non-reserved word>:

CHARACTERS

CODE_UNITS

NORMALIZE | NORMALIZED

OCTETS

4.5 Clause 5.3 <literal>

Add the following GR:

1.1) If the character set of a <character string literal> is UTF8, UTF16 or UTF32, then its value, *US*, is replaced by NORMALIZE (*US*).

4.6 Clause 6.1 <data type>

Modify Format as follows:

```
<length> ::= <unsigned integer> [<char length units>]

<large object length> ::=
   <unsigned integer> [ <multiplier> ] [<char length units>]
   | <large object length token> [<char length units>]
```

Add to Format:

```
<char length units> ::=
   OCTETS | CODE_UNITS | CHARACTERS
```

Note to proposal reader: The proposal to allow "GRAPHEMES" has been removed, at least until a stable and generally accepted definition is available for SQL to refer to.

Add GRs

1.1) If any specification or operation attempts to cause an item of a character type whose character set is UTF8, UTF16 or UTF32 to contain a code point that is a noncharacter, then an exception condition is raised: *data exception - noncharacter in UCS-string*.

Note to proposal reader: We are following the precedent set by GR1.

9.1) If <char length units> other than CHARACTERS is specified, then the conversion of the value of <length> to characters is implementation-defined.

Add a CR.

n) Without Feature Fnnn "UCS support", conforming SQL language shall not contain <char length units>.

4.7 Clause 6.18 <numeric value function>

Modify Format:

```
<char length expression> ::=
     { CHAR_LENGTH | CHARACTER_LENGTH }
          <left paren> <string value expression>
          [ USING <char length units> ] <right paren>

<string position expression> ::=
     POSITION <left paren> <string value expression>
        IN <string value expression> [ USING <char length units> ] <right paren>
```

Modify SR2 as follows:

2) If <string position expression> is specified, then both <string value expression>s shall be <bit value expression>s or both shall be <character value expression>s ~~having the same character repertoire~~ **that are comparable**.

Add an SR:

2.1) Case:

a) If the character set of a <character value expression> is not UTF8, UTF16 or UTF32, then <char length units> shall not be specified

b) Otherwise, if <char length units> is not specified, then CHARACTERS is implicit.

Note: We have specified CHARACTERS as the default because, in the only cases in which code units and code points are different (viz. UTF8 and UTF16), we believe users will only rarely wish to use CODE_UNITS.

Modify GR2 as follows:

2) If <string position expression> is specified, then

Case:

a) If the declared type of the two <string value expression>s is character, then let *SVE1* be the value of the first <string value expression> and let *SVE2* be the value of the second <string value expression>. If <char length units> is specified, then

Case:

i) If CHAR_LENGTH (*SVE1*) is 0 (zero), then the result is 1 (one).

ii) Let *CLU* be <char length units>. If there is at least one value *P* such that

***SVE2* = SUBSTRING (*SVE1* FROM *P* FOR CHAR_LENGTH (*SVE2* USING *CLU*))**

then the result is the least such *P*.

NOTE nnn - The collation used is determined in the normal way.

iii) Otherwise, the result is 0 (zero).

b) Otherwise

Case:

i) If the first <string value expression> has a length of 0 (zero), then the result is 1 (one).

ii) If the value of the first <string value expression> is equal to an identical-length substring of contiguous ~~characters or~~ bits from the value of the second <string value expression>, then the result is 1 (one) greater than the number of ~~characters or~~ bits within the value of the second <string value expression> preceding the start of the first such substring.

iii) Otherwise, the result is 0 (zero).

Note to proposal reader: The wording of subrule b) could be improved.

Modify GR5 as follows:

5) If a <char length expression> is specified, then

Case:

a) If the character set of <character value expression> is not UTF8, UTF16 or UTF32, then let S be the <string value expression>.

Case:

i) If the most specific type of S is character string, then the result is the number of characters in the value of S.

NOTE 100 - The number of characters in a character string is determined according to the semantics of the character set of that character string.

ii) Otherwise, the result is OCTET_LENGTH(S).

b) Otherwise, the result is the number of explicit or implicit <char length units> in <char length expression>, counted in accordance with the definition of those units in the relevant normatively referenced document.

Add a CR.

n) Without Feature Fnnn "UCS support", conforming SQL language shall not contain <char length units>.

4.8 Clause 6.19 <string value function>

Modify Format as follows:

```
<character substring function> ::=
     SUBSTRING <left paren> <character value expression>
     FROM <start position>
     [ FOR <string length> ] [ USING <char length units> ] <right paren>

<character overlay function> ::=
     OVERLAY <left paren> <character value expression>
     PLACING <character value expression>
     FROM <start position>
     [ FOR <string length> ] [ USING <char length units> ] <right paren>
```

Add to Format:

```
<normalize function> ::=
     NORMALIZE <left paren> <character value expression> <right paren>
```

Add an SR:

n+1) If <normalize function> is specified, then the declared type of the result is the declared type of <string value expression>.

Add a subrule to SR 4, beginning "If <character substring function> is specified, then"

b+1) If [USING <char length units>] is not specified, then USING CHARACTERS is implicit.

Note to proposal reader: We have specified CHARACTERS here as the default because, in the cases in which CODE_UNITS and CHARACTERS are different (viz. UTF8 and UTF16),
1. We believe users will only rarely wish to use CODE_UNITS, and
2. use of CODE_UNITS carries the danger of breaking a code point, so should be deliberate.

Modify GR3 by adding a subrule and modifying subrule a) as follows:

a-1) If the character set of <character value expression> is UTF8, UTF16 or UTF32, then, in the remainder of this General Rule, the term "character" shall be taken to mean "unit specified by <char length units>".

a) Let C be the value of the <character value expression>, let LC be the length **in characters** of C, and let S be the value of the <start position>.

Add a GR:

n) If <normalize function> is specified, then the result is the value of <string value expression>, in the normalized form of the result, in accordance with Unicode Standard Annex #15 Unicode Normalization Forms.

Note to proposal reader:
We believe no change is required to the following:
Clause 6.19 "<string value function>", General Rule 4) c):
If the length in characters of *E* ...(i.e. <escape character>) in <regular expression substring function> ...
Clause 6.19 "<string value function>", General Rule 8) d):
If the length in characters of *SC* ...(i.e. <trim character>)

Add a CR:

n) Without Feature Fnnn "UCS support", conforming SQL language shall not contain a <normalize function>.

n+1) Without Feature Fnnn "UCS support", conforming SQL language shall not contain <char length units>.

4.9 Clause 6.23 <cast specification>

Clause 6.23 "<cast specification>", General Rule 8) d) (casting to fixed-length character)

d) If *SD* is a fixed-length bit string or variable-length bit string, then let *LSV* be the value of BIT_LENGTH(*SV*) and let *B* be the BIT_LENGTH of the character with the smallest BIT_LENGTH in the ~~form-of-use~~ **character set** of *TD*. Let *PAD* be the value of the remainder of the division *LSV*/*B*. Let *NC* be a character whose bits all have the value 0 (zero).

4.10 Clause 6.28 <string value expression>

Modify SR 2) as follows:

2) **Case:**

a) If the character set of at least one character string in a <character value expression> is one of UTF8, UTF16 or UTF32, then the character set of every such character string shall be UTF8, UTF16 or UTF32.

b) Otherwise, ~~C~~character strings of different character repertoires shall not be mixed in a <character value expression>. The character repertoire of a <character value expression> is the character repertoire of its components.

Modify SR 4), beginning "4) Case: a) If <character factor> is specified, then", by adding the following subrule:

c) If <concatenation> is specified and the character set of <character factor> is UTF8, UTF16 or UTF32, then the character set of the result is specified by Subclause 4.2.n, "Universal character sets (UCS)".

Modify GR 2) as follows:

2) If <concatenation> is specified, then

a-1) If the character set of <character factor> is UTF8, UTF16 or UTF32, then, in the remainder of this General Rule, the term "length" shall be taken to mean "length in characters".

a) **~~I~~L**et *S1* and *S2* be the result of the <character value expression> and <character factor>, respectively.

b) Case:

~~a~~i) If either *S1* or *S2* is the null value, then the result of the <concatenation> is the null value.

~~b~~ii) Otherwise,

~~i~~1) **~~I~~L**et *S* be the string consisting of *S1* followed by *S2* and let *M* be the length of *S*.

2) If the character set of <character factor> is UTF8, UTF16 or UTF32, then *S* is replaced by:

Case:

A) If *S1* IS NORMALIZED AND *S2* IS NORMALIZED then NORMALIZE (*S*)

B) Otherwise, an implementation-defined string.

~~iii~~3) Case:

~~i~~**A**) If the most specific type of either *S1* or *S2* is large object character string, then let *LOL* be the implementation-defined maximum length of large object character strings.

Case:

~~1~~**I**) If *M* is less than or equal to *LOL*, then the result of the <concatenation> is *S* with length *M*.

~~2~~**II**) If *M* is greater than *LOL* and the right-most *M*-*LOL* characters of *S* are all the <space> character, then the result of the <concatenation> is the first *LOL* characters of *S* with length *LOL*.

~~3~~**III**) Otherwise, an exception condition is raised: data exception - string data, right truncation.

~~ii~~**B**) If the most specific type of either *S1* or *S2* is variable-length character string, then let *VL* be the implementation-defined maximum length of variable-length character strings.

Case:

~~1~~**I**) If *M* is less than or equal to *VL*, then the result of the <concatenation> is *S* with length *M*.

~~2~~**II**) If *M* is greater than *VL* and the right-most *M*-*VL* characters of *S* are all the <space> character, then the result of the <concatenation> the first *VL* characters of *S* with length *VL*.

~~3~~**III**) Otherwise, an exception condition is raised: data exception - string data, right truncation.

~~ii~~**C**) If the most specific types of both *S1* and *S2* are fixed-length character string, then the result of the <concatenation> is *S*.

Note to proposal reader: A Possible Problem regarding normalization of the result of concatenation is described in Section 6.1 "Normalization forms not closed under string concatenation", below.

Add a CR.

n) Without Feature Fnnn "UCS support", conforming SQL language shall not contain <char length units>.

4.11 Clause 8.n <normalized predicate>

Add a new subclause with the above title, as follows:

Format

```
<normalized predicate> ::=
   <string value expression> IS [ NOT ] NORMALIZED
```

Syntax Rules

1) The character set of <string value expression> shall be UTF8, UTF16 or UTF32.

Access Rules

None

General Rules

1) If the value of <string value expression> is null, then the result is *Unknown*.

2) If the value of <string value expression> is in Normalization Form C, as specified by Unicode Standard Annex #15 Unicode Normalization Forms, then the result is *True*, otherwise the result is *False*.

Conformance Rules

n) Without Feature Fnnn "UCS support", conforming SQL language shall not contain <normalized predicate >.

4.12 Clause 10.8 <collate clause>

Note to proposal reader: The proposal to allow <collation type> to specify the level of collation has been removed, because of the difficulty of specifying the meanings of the different levels.

Modify CR1) as follows:

1) Without **one or both of** Feature**s** F691, "Collation and translation" **and Fnnn "UCS support"**, conforming SQL language shall not contain any <collate clause>.

4.13 Clause 11.32 <collation definition>

This proposal is not concerned with language opportunities to allow a collation definition to specify that the collation is to be applicable to more than one character set, but we note that such an LO exists.

4.14 Clause 13.6 Data type correspondences

Table 19-Data type correspondences for C - Notes

Modify Notes [1] and [4] as follows:

[1]For character set UTF16, as well as other implementation-defined character sets in which ~~character elements~~ **a code unit** occup~~y~~**ies** two octets, k is the length in units of C unsigned short of the character encoded using the greatest number of such units in the character set; **for character set UTF32, as well as other implementation-defined character sets in which a code unit occupies four octets, k is four;** for other character sets, k is the length in units of C char of the character encoded using the greatest number of such units in the character set.

[4]For character set UTF16, as well as other implementation-defined character sets in which ~~character elements~~ **a code unit** occup~~y~~**ies** two octets, char or unsigned char should be replaced with unsigned short; **for character set UTF32, as well as other implementation-defined character sets in which a code unit occupies four octets,**

char or unsigned char should be replaced with unsigned int. Otherwise, char or unsigned char should be used.

Table 20-Data type correspondences for COBOL - Notes

Modify Note [4] as follows:

[4] For character set~~s UCS2 and~~ UTF16, as well as other implementation-defined character sets in which ~~character elements~~ **a code unit** occup~~y~~**ies** two octets, "PICTURE X(L)" should be replaced with "PICTURE N(L)". **For character set UTF32, as well as other implementation-defined character sets in which a code unit occupies four octets, "PICTURE X(L)" should be replaced with "PICTURE ????".** Otherwise, "PICTURE X(L)" should be used.

Note to proposal reader: We have been unable to discover what should replace "????" in the foregoing. Help is requested.

Table 21-Data type correspondences for Fortran - Notes

Modify Note [3] as follows:

[3] For character sets ~~UCS2 and~~ UTF16 **and UTF32**, as well as other implementation-defined character sets in which ~~character elements~~ **a code unit** occup~~y~~**ies** ~~two octets~~ **more than one octet**, "CHARACTER KIND=n" should be used; in this case, the value of n that corresponds to a given character set is implementation-defined. Otherwise, "CHARACTER" (without "KIND=n") should be used.

4.15 Clause 13.4 Calls to an <externally-invoked procedure>

Modify Syntax Rule2) e), by inserting the following:

```
DATA_EXCEPTION_NONCHARACTER_IN_UCS_STRING:
     constant SQLSTATE_TYPE :="22nnn";
```

4.16 Clause 23.1 SQLSTATE

Table 34 "SQLSTATE class and subclass values"

Add the following row:

Category	Condition	Class	Subcondition	Subclass
X	**data exception**	**22**	***noncharacter in UCS-string***	**nnn**

4.17 Clause 24.1 General conformance requirements

Add the following paragraph:

Note to proposal reader: The point at which this should be added is left to the editor's discretion.

An SQL-implementation that claims conformance to Feature 471 UCS support shall

- **conform to ISO/IEC 10646-1:2000 at some specified level;**
- **provide at least one of the named character sets UTF8, UTF16 and UTF32;**

- **provide, as the default collation for each such character set, a collation that conforms to ISO/IEC 14651:2001 at some level.**

Table 36-Implied feature relationships

Add the following row:

Feature ID	Feature Description	Implied Feature ID	Implied Feature Description
F471	**UCS support**	**F461**	**Named character sets**

4.18 Annex B (informative) Implementation-defined elements

Add one item:

2.1) Subclause 4.2.5 Universal character sets (UCS)

a) With the exception of <normalize function> and <normalized predicate>, the result of any operation on an unnormalized UCS string is implementation-defined.

Modify one item as follows:

28) Clause 6.28 <string value expression>

a) If the result of the <character value expression> is a zero-length character string, then it is implementation-defined whether an exception condition is raised: data exception -zero-length character string.

b) If the character set of <character factor> is UTF8, UTF16 or UTF32, and either of the operands is not normalized, then the result is implementation-defined.

4.19 Annex F (informative) SQL feature and package taxonomy

Table 38-Feature taxonomy for features outside Core SQL

Add the following row

	Feature ID	Feature Name
nnn	**F471**	**UCS support**

5 Possible Problems to be deleted

PPs 691 and 774, relating to Counting Characters.

Note that PP 691 was added by CWB-051, which said:

> Add the following Editor's Note (to be resolved before the end of the SQL3 FCD Editing Meeting!) somewhere associated with the CHARACTER_LENGTH function:
>
> ** Editor's Note **
>
> There is a possible problem with the interaction between the CHARACTER_LENGTH function and the use of Unicode as a character paradigm. Consider the glyph "á". In Unicode, this is canonically decomposed as the two Unicode characters "a" followed by "" (non-spacing acute accent), but there exists in Unicode the "composed" character "á" as well. What is the value of
>
> CHARACTER_LENGTH('á')? Does it differ if the underlying character set is Latin1 versus Unicode?

Language Opportunities 775 (Collations) and 776 (Normalization)

6 Comments to be closed

#95 NLD-P02-022, #306 GBR-P02-500, #297 NLD-P02-182, #410 NLD-P02-184, #25 USA-P02-001

7 Possible Problems identified

7.1 Normalization forms not closed under string concatenation

Add the following Possible Problem

Severity: Major Technical

Reference: P02, SQL/Foundation, Clause 6.28 <string value expression>.

Note at: None.

Source: WG3: PER-054

Possible Problem:

The following is a quote from Unicode Standard Annex #15 "Unicode Normalization Forms":

> None of the normalization forms are closed under string concatenation. Consider the following examples:

Form	String1	String2	Concatenation	Correct Normalization
NFC	"a"	"^"	"a"+"^"	"â"
NFD	"a"+"^"	"." (dot under)	"a"+"^" + "."	"a" + "." +"^"

> Without limiting the repertoire, there is no way to produce a normalized form that is closed under simple string concatenation. If desired, however, a specialized function could be constructed that produced a normalized concatenation.

The approach of PER-054 is based on the principle of early uniform normalization. The user is responsible for normalizing data and is provided with tools to help. The only operation that may result in loss of normalization is concatenation. It is therefore desirable to specify the result of concatenation so that such loss is avoided. This offers the user integrity, at small effort to the implementor. Moreover, the performance cost is much less than the cost of completely renormalizing the result of every concatenation, as a cautious user might feel obliged to do.

This is because, if the first character of String2 is not a combining character, then no action at all is required. Otherwise, the only part of the result that requires treatment is the substring whose first character is the last non-combining character of String1 and whose last character is the combining character that immediately precedes the first non-combining character in String2.

The problem is: how to specify a normalization-preserving concatenation, so that there is no unnecessary performance penalty, and yet the result is not non-deterministic.
The approach attempted was to add a subrule to GR 2 of Clause 6.28 <string value expression>, whose effect is to require the result to be:

> Case:
>
> If *S1* IS NORMALIZED AND *S2* IS NORMALIZED then NORMALIZE (*S*)
>
> otherwise implementation-defined.

This condition can be satisfied by taking the action described above, without regard to whether *S1* or *S2* is normalized.

Solution

None provided with comment.

8 Checklist

Concepts	**Y**
Access Rules	Y
Conformance Rules	Y
Lists of SQL-statements by category	n/a
Table of identifiers used by diagnostic statements	n/a
Collation coercibility for character strings	n/a
Closing Possible Problems	Y
Any new Possible Problems clearly identified	Y
Reserved and non-reserved keywords	Y
SQLSTATE tables and Ada package	Y
Information and Definition Schemas	see [PER-095]
Implementation-defined and -dependent Annexes	n/a
Incompatibilities Annex	n/a
Embedded SQL and host language implications	n/a
Dynamic SQL issues: including descriptor areas	n/a
CLI issues	n/a

***** End of paper *****