INTERNATIONAL STANDARD ISO/IEC 9075-2:1999
TECHNICAL CORRIGENDUM 2

Published 2003-06-01

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION • МЕЖДУНАРОДНАЯ ОРГАНИЗАЦИЯ ПО СТАНДАРТИЗАЦИИ • ORGANISATION INTERNATIONALE DE NORMALISATION
INTERNATIONAL ELECTROTECHNICAL COMMISSION • МЕЖДУНАРОДНАЯ ЭЛЕКТРОТЕХНИЧЕСКАЯ КОМИССИЯ • COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

Information technology — Database languages — SQL —

Part 2:
Foundation (SQL/Foundation)

TECHNICAL CORRIGENDUM 2

Technologies de l'information — Langages de base de données — SQL —

Partie 2: Fondations (SQL/Fondations)

RECTIFICATIF TECHNIQUE 2

Technical Corrigendum 2 to ISO/IEC 9075-2:1999 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*. ISO/IEC 9075-2:1999/Cor. 2:2003 cancels and replaces ISO/IEC 9075-2:1999/Cor. 1:2000.

Statement of purpose for rationale:

A statement indicating the rationale for each change to ISO/IEC 9075 is included. This is to inform the users of that standard as to the reason why it was judged necessary to change the original wording. In many cases the reason is editorial or to clarify the wording; in some cases it is to correct an error or an omission in the original wording.

Notes on numbering:

Where this Corrigendum introduces new Syntax, Access, General and Conformance Rules, the new rules have been numbered as follows:
Rules inserted between, for example, Rules 7) and 8) are numbered 7.1), 7.2), etc. [or 7) a.1), 7) a.2), etc.]. Those inserted before Rule 1) are numbered 0.1), 0.2), etc.
Where this Corrigendum introduces new Subclauses, the new subclauses have been numbered as follows:
Subclauses inserted between, for example, Subclause 4.3.2 and 4.3.3 are numbered 4.3.2a, 4.3.2b, etc.
Those inserted before, for example, 4.3.1 are numbered 4.3.0, 4.3.0a, etc.

ICS 35.060 **Ref. No. ISO/IEC 9075-2:1999/Cor.2:2003(E)**

Published in Switzerland

Contents

Page

(Blank page)

Information technology — Database languages — SQL —

Part 2:
Foundation (SQL/Foundation)

TECHNICAL CORRIGENDUM 2

2 Normative references

1. Rationale: Add missing references..

Insert the following items to this Subclause:

ISO/IEC 8859-1:1998, Information technology — 8-bit single-byte coded graphic character sets - Part 1: Latin alphabet No. 1.

ISO/IEC 6429:1992, Information technology — Control functions for coded character sets.

The Internet Assigned Numbers Authority, Character sets.
http://www.iana.org/assignments/character-sets

3.1.1 Definitions taken from ISO/IEC 10646

1. Rationale: Remove unused definitions.

Replace the Subclause with:

This part of ISO/IEC 9075 makes use of the following terms defined in ISO/IEC 10646:

a) character
NOTE 0.1 – This is identical to the Unicode definition of *abstract character*. In ISO/IEC 9075, when the relevant character repertoire is UCS, a character can be thought of as *that which is represented by one code point*.

b) repertoire

3.1.2 Definitions taken from Unicode

1. Rationale: Remove unused definitions.

Replace the Subclause with:

This part of ISO/IEC 9075 makes use of the following terms defined in The Unicode Standard:

a) character encoding form

b) code point

c) control character

d) transcoding

3.1.5 Definitions provided in Part 2

1. Rationale: Clarify assignable and comparable.

Replace item a) with:

a) **assignable**: (of types, taken pair wise) The characteristic of a data type, T1, that permits a value of T1 to be assigned to a site of a specified data type, T2 (where T1 may be the same as T2).

2. Rationale: Clarify the definition of "assignment"

Replace item b) with:

b) **assignment**: The operation whose effect is to ensure that the value at a site *T* (known as the target) is identical to a given value *S* (known as the source). Assignment is frequently indicated by the use of the phrase *"T* is set to *S*" or "the value of *T* is set to *S*".

3. Rationale: Clarify the definition of "comparable"

Replace item i) with:

i) **comparable** (of a pair of values): Capable of being compared, according to the rules of Subclause 8.2 "<comparison predicate>". In most, but not all, cases the values of a data type can be compared one with another. For the specification of comparability of individual data types, see Subclauses 4.2 to 4.11.

4. Rationale: Correct and clarify the notion of "distinct".

Replace item l) with:

l) **distinct**: (of a pair of comparable values): Informally: not equal or not both null or having a pair of corresponding components that are distinct. Formally:

Two null values of comparable type are not distinct.

A null value and a non-null value of comparable type are distinct.

For two non-null values, the following rules apply:

— Two values of predefined type or reference type are distinct if and only if they are not equal.

— If two values *V1* and *V2* are of a user-defined type whose comparison form is RELATIVE or MAP and the result of comparing them for equality according to Subclause 8.2, "<comparison predicate>" is *Unknown*, then it is implementation-dependent whether they are distinct or not; otherwise, they are distinct if and only if they are not equal.

— If two values *V1* and *V2* are of a structured type whose comparison form is STATE, then they are distinct if their most specific types are different, or if there is an attribute *A* of their common most specific type such that the value of *A* in *V1* and the value of *A* in *V2* are distinct.

— Two rows (or partial rows) are distinct if and only if at least one of their pairs of respective fields is distinct.

— Two arrays are distinct if the arrays do not have the same cardinality or if, for arrays of the same cardinality, elements in the same ordinal position in the two arrays are distinct.

NOTE 0.1 — The result of evaluating whether or not two comparable values are distinct is never *Unknown*. The result of evaluating whether or not two values that are not comparable, for example, values of a user-defined type that has no comparison type, is not defined.

5. *Rationale: Define "equals".*

Insert the following definition:

o.1) **equal** (of a pair of comparable values): Yielding *True* if passed as arguments in a <comparison predicate> in which the <comp op> is <equals operator>. See Subclause 8.2, "<comparison predicate>".

6. *Rationale: Define "identical"*

Insert the following definition:

s.1) **identical** (of a pair of values): Indistinguishable, in the sense that it is impossible, by any means available in SQL, to detect any difference between them. For the full definition, see Subclause 9.0 "Determination of identical values".

7. *Rationale: Clarify the distinction between character sets and character repertoires.*

Replace item rr) with:

rr) **transliteration**: A method of translating characters in one character set into characters of the same or a different character set.

8. *Rationale: Remove <identifier ignorable character>s from the definition of <white space>.*

In item vv) delete the following bullet items:

— U+200C, Zero Width Non-Joiner

— U+200D, Zero Width Joiner

— U+200E, Left-To-Right Mark

— U+200F, Right-To-Left Mark

9. *Rationale: Enhance the definition of <white space> for implementations supporting Unicode 3.0.*

In item vv) insert the following bullet item:

— U+202F, Narrow No-Break Space

10. Rationale: Remove Zero Width Space from the definition of <white space>

In item vv), delete the following bullet item:

— U+200B, Zero Width Space

3.3.1.2 Other terms

1. Rationale: Add definition of direct result of executing an SQL-statement.

Insert the following Subclause:

3.3.1.2 Other terms

An SQL-statement *S1* may be said to be executed as a *direct result of executing an SQL-statement* if *S1* is the SQL-statement contained in an <externally-invoked procedure> or <SQL-invoked routine> that has been executed.

4.1 Data types

1. Rationale: Change "user-defined data type" to "user-defined type".

Replace the 3rd paragraph with:

SQL supports three sorts of data types: *predefined data types*, *constructed types*, and *user-defined types*. Predefined data types are sometimes called the "built-in data types", though not in this International Standard. User-defined types can be defined by a standard, by an implementation, or by an application.

2. Rationale: Correct oversight in definition of directly based on.

Replace the 11th paragraph with:

A structured type *ST* is *directly based on* a data type *DT* if any of the following are true:

a) *DT* is the declared type of some attribute of *ST*.

b) *DT* is the direct supertype of *ST*.

c) *DT* is a direct subtype of *ST*.

d) *DT* is compatible with *ST*.

3. Rationale: Add definition of non-referentially based on.

Insert the following paragraph immediately after the 15th paragraph:

A data type *DT1* is non-referentially based on a data type *DT2* if *DT1* is not a reference type and is either directly based on *DT2* or is directly based on some data type that is non-referentially based on *DT2*.

4. Rationale: Define "usage-dependent" for use in Access Rules

Insert the following inserted paragraph immediately before the 16th paragraph:

A type *TY* is *usage-dependent* on a user-defined type *UDT* if one of the following conditions is true:

— *TY* is *UDT*;

— *TY* is a reference type whose referenced type is *UDT*;

— *TY* is a row type, and the declared type of a field of *TY* is usage-dependent on *UDT*; or

— *TY* is a collection type, and the declared element type of *TY* is usage-dependent on *UDT*.

5. Rationale: Provide consistent rules for comparison operations.

Insert the following paragraphs after the last paragraph:

Ordering and comparison of values of the predefined data types requires knowledge only about those predefined data types. However, to be able to compare and order values of constructed types or of user-defined types, additional information is required. We say that some type *T* is *S*-ordered, for some set of types *S*, if, in order to compare and order values of type *T*, information about ordering at least one of the types in *S* is first required. A definition of *S*-ordered is required for several *S* (that is, for several sets of types), but not for all possible such sets.

The general definition of *S*-ordered is this:

Let *T* be a type and let *S* be a set of types. Then *T* is *S*-ordered if one of the following is true:

1. *T* is a member of *S*.
2. *T* is a row type and the declared type of some field of *T* is *S*-ordered.
3. *T* is an array type and the element type of *T* is *S*-ordered.
4. *T* is a structured type whose comparison form is STATE and the declared type of some attribute of *T* is *S*-ordered.
5. *T* is a user-defined type whose comparison form is MAP and the return type of the SQL-invoked function that is identified by the <map function specification> is *S*-ordered.
6. *T* is a reference type with a derived representation and the declared type of some attribute enumerated by the <derived representation> is *S*-ordered.

The notion of *S*-ordered is applied in the following definitions:

A type *T* is *LOB-ordered* if *T* is *L*-ordered, where *L* is the set of large object types.

A type *T* is *array-ordered* if *T* is *ARR*-ordered, where *ARR* is the set of array types.

A type *T* is *reference-ordered* if *T* is *REF*-ordered, where *REF* is the set of reference types.

A type *T* is *DT-EC-ordered* if *T* is *DTE*-ordered, where *DTE* is the set of distinct types with EQUALS ONLY comparison form (DT-EC stands for "distinct type - equality comparison")

A type *T* is *DT-FC-ordered* if *T* is *DTF*-ordered, where *DTF* is the set of distinct types with FULL comparison form.

A type *T* is *DT-NC-ordered* if *T* is *DTN*-ordered, where *DTN* is the set of distinct types with no comparison form.

A type *T* is *ST-EC-ordered* if *T* is *STE*-ordered, where *STE* is the set of structured types with EQUALS ONLY comparison form.

A type *T* is *ST-FC-ordered* if *T* is *STF*-ordered, where *STF* is the set of structured types with FULL comparison form.

A type *T* is *ST-NC-ordered* if *T* is *STN*-ordered, where *STN* is the set of structured types with no comparison form.

A type *T* is *ST-ordered* if *T* is either ST-EC-ordered, ST-FC-ordered or ST-NC-ordered.

A type *T* is *UDT-EC-ordered* if *T* is either DT-EC-ordered or ST-EC-ordered (UDT stands for "user-defined type").

A type *T* is *UDT-FC-ordered* if *T* is either DT-FC-ordered or ST-FC-ordered

A type *T* is *UDT-NC-ordered* if T is either DT-NC-ordered or ST-NC-ordered

6) *Rationale: Definition required to define certain kinds of non-determinism involving comparison of datetime with time zone with datetime without time zone.*

Insert the following paragraph after the last paragraph:

The notion of *constituent* of a declared type *DT* is defined recursively as follows:

— *DT* is a constituent of *DT*.

— If *DT* is a row type, then each field of *DT* is a constituent of *DT*

— If *DT* is a collection type, then the element type of *DT* is a constituent of *DT*.

— Every constituent of a constituent of *DT* is a constituent of *DT*.

7) *Rationale: Clarify assignable and comparable.*

Insert the following paragraph after the last paragraph:

Two data types, *T1* and *T2*, are said to be *compatible* if *T1* is assignable to *T2*, *T2* is assignable to *T1*, and their descriptors include the same data type name. If they are row types, it must further be the case that the declared types of their corresponding fields are pairwise compatible. If they are collection types, it must further be the case that their element types are compatible. If they are reference types, it must further be the case that their referenced types are compatible.

NOTE 1.1 — The data types "CHARACTER(*n*) CHARACTER SET *CS1*" and "CHARACTER(*m*) CHARACTER SET *CS2*", where *CS1* ≠ *CS2*, have descriptors that include the same data type name (CHARACTER), but are not mutually assignable; therefore, they are not compatible.

4.2 Character strings

1. Rationale: Clarify the distinction between character sets and character repertoires.

Replace the 1st and 2nd paragraphs with:

A character string is a sequence of characters. All the characters in a character string are taken from a single character set. A character string has a length, which is the number of characters in the sequence. The length is 0 (zero) or a positive integer.

A character type is described by a character type descriptor. A character type descriptor contains:

— The name of the specific character type (CHARACTER, CHARACTER VARYING, and CHARACTER LARGE OBJECT; NATIONAL CHARACTER, NATIONAL CHARACTER VARYING, and NATIONAL CHARACTER LARGE OBJECT are represented as CHARACTER, CHARACTER VARYING, and CHARACTER LARGE OBJECT, respectively).

— The length or maximum length in characters of the character type.

— The catalog name, schema name, and character set name of the character set of the character type.

— The catalog name, schema name, and collation name of the collation of the character type.

The character set of a character type may be specified explicitly or implicitly.

The <key word>s NATIONAL CHARACTER are used to specify an implementation-defined character set. Special syntax (N'string') is provided for representing literals in that character set.

With two exceptions, a character string expression is assignable only to sites of a character type whose character set is the same. The exceptions are as specified in Subclause 4.2.4, "Universal character sets", and such other cases as may be implementation-defined. If a store assignment would result in the loss of non-<space> characters due to truncation, then an exception condition is raised. If a retrieval assignment or evaluation of a <cast specification> would result in the loss of characters due to truncation, then a warning condition is raised.

Character sets fall into three categories: those defined by national or international standards, those defined by SQL-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 Subclause 4.2.4, "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 SQL-implementations reside in the Information Schema (named INFORMATION_SCHEMA) in each catalog, as do collations defined by standards and collations, transliterations, and transcodings defined by SQL-implementations.

4.2.1 Character strings and collating sequences

1. *Rationale: Remove redundant definition. Clarify the distinction between character sets and character repertoires.*

Replace the entire subclause with:

4.2.1 Comparison of character strings

Two character strings are comparable if and only if either they have the same character set or there exists at least one collation that is applicable to both their respective character sets.

A collation is defined by ISO/IEC 14651 as "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". Each collation known in an SQL-environment is applicable to one or more character sets, and for each character set, one or more collations are applicable to it, one of which is associated with it as its *character set collation*.

Anything that has a declared type can, if that type is a character type, be associated with a collation applicable to its character set; this is known as a declared type collation. Every declared type that is a character type has a collation derivation, this being either none, implicit, or explicit. The collation derivation of a declared type with a declared type collation that is explicitly or implicitly specified by a <data type> is implicit. If the collation derivation of a declared type that has a declared type collation is not implicit, then it is explicit. The collation derivation of an expression of character type that has no declared type collation is none.

An operation that explicitly or implicitly involves character string comparison is a character comparison operation. At least one of the operands of a character comparison operation must have a declared type collation.

There may be an SQL-session collation for some or all of the character sets known to the SQL-implementation (see Subclause 4.39, "SQL-sessions").

The collation used for a particular comparison is determined as in Subclause 4.2.3, “Rules determining collating sequence usage”.

The comparison of two character string expressions depends on the collation used for the comparison. When values of unequal length are compared, if the collation for the comparison has the NO PAD characteristic and the shorter value is equal to some prefix of the longer value, then the shorter value is considered less than the longer value. If the collation for the comparison has the PAD SPACE characteristic, for the purposes of the comparison, the shorter value is effectively extended to the length of the longer by concatenation of <space>s on the right.

For every character set, there is at least one collation.

4.2.2.1 Operators that operate on character strings and return character strings

1. *Rationale: Clarify that the result of case folding may be of a different length in characters than the source string.*

Insert the following paragraph after the 4th paragraph:

NOTE 2.1 — Case correspondences are not always one-to-one: the result of case folding may be of a different length in characters than the source string. For example, U+00DF ß LATIN SMALL LETTER SHARP S becomes "SS" in upper case.

4.2.4 Named character sets

1. *Rationale: Clarify the distinction between character sets and character repertoires.*

Replace the subclause with:

4.2.3a Character repertoires

An SQL-implementation supports one or more character repertoires. These character repertoires may be defined by a standard or be implementation-defined.

A character repertoire is described by a character repertoire descriptor. A character repertoire descriptor includes:

– The name of the character repertoire.

– The name of the default collation for the character repertoire.

The following character repertoire names are specified as part of ISO/IEC 9075:

– SQL_CHARACTER is a character repertoire that consists of the 88 <SQL language character>s as specified in Subclause 5.1, "<SQL terminal character>". The name of the default collation is SQL_CHARACTER.

– GRAPHIC_IRV is the character repertoire that consists of the 95-character graphic subset of the International Reference Version (IRV) as specified in ISO 646:1991. Its repertoire is a proper superset of that of SQL_CHARACTER. The name of the default collation is GRAPHIC_IRV.

– LATIN1 is the character repertoire defined in ISO 8859-1. The name of the default collation is LATIN1.

– ISO8BIT is the character repertoires formed by combining the character repertoire specified by ISO 8859-1 and the "control characters" specified by ISO 6429. The repertoire consists of all 255 characters, each consisting of exactly 8 bits, as, including all control characters and all graphic characters except the character corresponding to the numeric value 0 (zero). The name of the default collation is ISO8BIT.

– UCS is the character repertoire is the Universal Character Set repertoire specified by The Unicode Standard Version 3.1 and by ISO/IEC 10646. It is implementation-defined whether the name of the default collation is UCS_BASIC or UNICODE.

– SQL_TEXT is a character repertoire that is an implementation-defined subset of the repertoire of the Universal Character Set that includes every <SQL language character> and every character in every character set supported by the SQL-implementation. The name of the default collation is SQL_TEXT.

– SQL_IDENTIFIER is a character repertoire consisting of the <SQL language character>s and all other characters that the SQL-implementation supports for use in <regular identifier>s. The name of the default collation is SQL_IDENTIFIER.

4.2.3b Character encoding forms

An SQL-implementation supports one or more character encoding forms for each character repertoire that it supports. These character encoding forms may be defined by a standard or be implementation-defined.

A character encoding form is described by a character encoding form descriptor. A character encoding form descriptor includes:

– The name of the character encoding form.

– The name of the character repertoire to which it is applicable.

The following character encoding form names are specified as part of ISO/IEC 9075:

– SQL_CHARACTER is an implementation-defined character encoding form. It is applicable to the SQL_CHARACTER character repertoire.

– GRAPHIC_IRV is the character encoding form in which the coded representation of each character is specified in ISO 646:1991. It is applicable to the GRAPHIC_IRV character repertoire.

– LATIN1 is the character encoding form specified in ISO 8859-1. It is applicable to the LATIN1 character repertoire.

– ISO8BIT is the character encoding form specified in ISO 8859-1, augmented by ISO 6429. When restricted to the LATIN1 characters, it is the same character encoding form as LATIN1. It is applicable to the ISO8BIT character repertoire.

– UTF32 is the character encoding form specified in the Unicode Standard Annex #19, "UTF-32", in which each character is encoded as four octets. It is applicable to the UCS character repertoire.

– UTF16 is the character encoding form specified in ISO/IEC 10646-1, Annex C (normative), "Transformation format for 16 planes of Group 00 (UTF-16)", in which each character is encoded as two or four octets. It is applicable to the UCS character repertoire.

– UTF8 is the character encoding form specified in ISO/IEC 10646-1, Annex D (normative), "UCS Transformation Format 8 (UTF-8)", in which each character is encoded as from one to four octets. It is applicable to the UCS character repertoire.

– SQL_TEXT is an implementation-defined character encoding form. It is applicable to the SQL_TEXT character repertoire.

– SQL_IDENTIFIER is an implementation-defined character encoding form. It is applicable to the SQL_IDENTIFIER character repertoire.

If an SQL-implementation supplies more than one character encoding form for a particular character repertoire, then it shall specify a precedence ordering of the character encoding forms of that character repertoire. The precedence of character encoding forms applicable to the UCS character repertoire and defined in this part of ISO/IEC 9075 is:

UTF32 ≺ UTF16 ≺ UTF8

4.2.3c Collations

An SQL-implementation supports one or more collations for each character encoding form that it supports.

A collation is described by a collation descriptor. A collation descriptor includes:

— The name of the collation.

— The name of the character repertoire to which it is applicable.

— A list of the names of the character sets to which the collation can be applied.

— Whether the collation has the NO PAD or the PAD SPACE characteristic.

The following SQL supported collation names are specified as part of ISO/IEC 9075:

— SQL_CHARACTER is an implementation-defined collation. It is applicable to the SQL_CHARACTER character repertoire.

— GRAPHIC_IRV is a collation in which the ordering is determined by treating the code points defined by ISO 646:1991 as unsigned integers. It is applicable to the GRAPHIC_IRV character repertoire.

— LATIN1 is a collation in which the ordering is determined by treating the code points defined by ISO 8859-1 as unsigned integers. It is applicable to the LATIN1 character repertoire.

— ISO8BIT is a collation in which the ordering is determined by treating the code points defined by ISO 8859-1 as unsigned integers. When restricted to the LATIN1 characters, it produces the same collation as LATIN1. It is applicable to the ISO8BIT character repertoire.

— UCS_BASIC is a collation in which the ordering is determined entirely by the Unicode scalar values of the characters in the strings being sorted. It is applicable to the UCS character repertoire.
NOTE 2.2 - The Unicode scalar value of a character is its code point treated as an unsigned integer.

— UNICODE is a collation in which the ordering is determined by Unicode Technical Standard #10, Unicode Collation Algorithm, Version 8.0. It is applicable to the UCS character repertoire.

— SQL_TEXT is an implementation-defined collation. It is applicable to the SQL_TEXT character repertoire.

— SQL_IDENTIFIER is an implementation-defined collation. It is applicable to the SQL_IDENTIFIER character repertoire.

4.2.4 Character sets

An SQL <character set specification> allows a reference to a character set name defined by a standard, an SQL-implementation, or a user.

A character set is described by a character set descriptor. A character set descriptor includes:

— The name of the character set.

— The name of the character repertoire for the character set.

— The name of the character encoding form for the character set.

— The name of the default collation for the character set.

The following SQL supported character set names are specified as part of ISO/IEC 9075:

— SQL_CHARACTER is a character set whose repertoire is SQL_CHARACTER and whose character encoding form is SQL_CHARACTER. The name of the default collation is SQL_CHARACTER.

— GRAPHIC_IRV is a character set whose repertoire is GRAPHIC_IRV and whose character encoding form is GRAPHIC_IRV. The name of the default collation is GRAPHIC_IRV.

— ASCII_GRAPHIC is a synonym for GRAPHIC_IRV.

— LATIN1 is a character set whose repertoire is LATIN1 and whose character encoding form is LATIN1. The name of the default collation is LATIN1.

— ISO8BIT is a character set whose repertoire is ISO8BIT and whose character encoding form is ISO8BIT. The name of the default collation is ISO8BIT.

— ASCII_FULL is a synonym for ISO8BIT.

— UTF32 is a character set whose repertoire is UCS and whose character encoding form is UTF32. It is implementation-defined whether the name of the default collation is UCS_BASIC or UNICODE.

— UTF16 is a character set whose repertoire is UCS and whose character encoding form is UTF16. It is implementation-defined whether the name of the default collation is UCS_BASIC or UNICODE.

— UTF8 is a character set whose repertoire is UCS and whose character encoding form is UTF8. It is implementation-defined whether the name of the default collation is UCS_BASIC or UNICODE.

— SQL_TEXT is a character set whose repertoire is SQL_TEXT and whose character encoding form is SQL_TEXT. The name of the default collation is SQL_TEXT.

— SQL_IDENTIFIER is a character set whose repertoire is SQL_IDENTIFIER and whose character encoding form is SQL_IDENTIFIER. The name of the default collation is SQL_IDENTIFIER.

The result of evaluating a character string expression whose most specific type has character set *CS* is constrained to consist of characters drawn from the character repertoire of *CS*.

Character set names are registered with the Internet Assigned Names Authority (IANA), at http://www.iana.org/assignments/character-sets.

Table 3.1 – Overview of character sets

Character Set	Character Repertoire	Character Encoding Form	Collation	Synonym
GRAPHIC_IRV	GRAPHIC_IRV	GRAPHIC_IRV	GRAPHIC_IRV	ASCII_GRAPHIC
ISO8BIT	ISO8BIT	ISO8BIT	ISO8BIT	ASCII_FULL
LATIN1	LATIN1	LATIN1	LATIN1	
SQL_CHARACTER	SQL_CHARACTER	SQL_CHARACTER	SQL_CHARACTER	
SQL_TEXT	SQL_TEXT	SQL_TEXT	SQL_TEXT	
SQL-IDENTIFIER	SQL-IDENTIFIER	SQL-IDENTIFIER	SQL-IDENTIFIER	
UTF16	UCS	UTF16	UCS_BASIC or UNICODE	
UTF32	UCS	UTF32	UCS_BASIC or UNICODE	
UTF8	UCS	UTF8	UCS_BASIC or UNICODE	

NOTE 2.3 – An SQL-implementation may supply additional character sets and/or additional character encoding forms and collations for character sets defined in this Part of ISO/IEC 9075.

4.2.4a Universal character sets

A *UCS string* is a character string whose character set whose descriptor names both the character repertoire UCS and one of the character encoding forms 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. 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.

4.3 Binary strings

1. Rationale: Clarify assignable and comparable.

Insert the following paragraph after the 2nd paragraph:

A binary string is assignable only to sites of data type BINARY LARGE OBJECT. If a store assignment would result in the loss of non-zero octets due to truncation, then an exception condition is raised. If a retrieval assignment would result in the loss of octets due to truncation, then a warning condition is raised.

4.3.1 Binary string comparison

1. Rationale: Remove redundant definition. Clarify assignable and comparable.

Replace the paragraph with:

All binary string values are comparable. When binary string values are compared, they must have exactly the same length (in octets) to be considered equal. Binary string values can be compared only for equality.

4.4.1 Bit string comparison and assignment

1. Rationale: Remove redundant definition.

Replace the paragraph with:

All bit string values are comparable.

If a store assignment would result in the loss of bits due to truncation, then an exception condition is raised. If a store assignment to a fixed-length bit string would result in the addition of bits, then an exception condition is raised. If a retrieval assignment would result in the loss of bits due to truncation, then a warning condition is raised. When values of unequal length are compared, if the shorter is a prefix of the longer, then the shorter is less than the longer; otherwise, the longer is effectively truncated to the length of the shorter for the purposes of comparison. When values of equal length are compared, then a bit-by-bit comparison is made. A 0-bit is less than a 1-bit.

4.5 Numbers

1. Rationale: Clarify assignable and comparable.

Replace the 1st, 2nd and 3rd paragraphs with:

A number is either an exact numeric value or an approximate numeric value. Any two numbers are comparable.

A numeric type is described by a numeric type descriptor. A numeric type descriptor contains:

— The name of the specific numeric type (NUMERIC, DECIMAL, SMALLINT, INTEGER, BIGINT, FLOAT, REAL, or DOUBLE PRECISION).

— The precision of the numeric type.

— The scale of the numeric type, if it is an exact numeric type.

— An indication of whether the precision (and scale) are expressed in decimal or binary terms

An SQL-implementation is permitted to regard certain <exact numeric type>s as equivalent, if they have the same precision, scale, and radix, as permitted by the Syntax Rules of Subclause 6.1, "<data type>". When two or more <exact numeric type>s are equivalent, the SQL-implementation chooses one of these equivalent <exact numeric type>s as the normal form representing that equivalence class of <exact numeric type>s. The normal form determines the name of the exact numeric type in the numeric type descriptor.

Similarly, an SQL-implementation is permitted to regard certain <approximate numeric type>s as equivalent, as permitted the Syntax Rules of Subclause 6.1, "<data type>", in which case the SQL-implementation chooses a normal form to represent each equivalence class of <approximate numeric type> and the normal form determines the name of the approximate numeric type.

For every numeric type, the least value is less than zero and the greatest value is greater than zero.

4.5.1 Characteristics of numbers

1. *Rationale: Clarify assignable and comparable.*

Replace the 1st paragraph with:

An exact numeric type has a precision, *P*, and a scale, *S*. *P* is a positive integer that determines the number of significant digits in a particular radix, *R*, where *R* is either 2 or 10. *S* is a non-negative integer. Every value of an exact numeric type of scale *S* is of the form $n*10^{-S}$, where *n* is an integer such that $-(R^P) \le n < RP$.
NOTE 4.1 — Not every value in that range is necessarily a value of the type in question.

2. *Rationale: Clarify numeric precision.*

Replace the 2nd paragraph with:

An approximate numeric value consists of a mantissa and an exponent. The mantissa is a signed numeric value, and the exponent is a signed integer that specifies the magnitude of the mantissa. An approximate numeric value has a precision. The precision is a positive integer that specifies the number of significant binary digits in the mantissa. The value of an approximate numeric value is the mantissa multiplied by a factor determined by the exponent.

An <approximate numeric literal> *ANL* consists of an <exact numeric literal> (called the <mantissa>), the letter 'E' or 'e', and a <signed integer> (called the <exponent>). If *M* is the value of the <mantissa> and *E* is the value of the <exponent>, then $M * 10^E$ is the *apparent value* of *ANL*. The actual value of *ANL* is approximately the apparent value of *ANL*, according to implementation-defined rules.

3. *Rationale: Clarify assignable and comparable.*

Insert the following paragraph after the 2nd paragraph with:

A number is assignable only to sites of numeric type. If an assignment of some number would result in a loss of its most significant digit, an exception condition is raised. If least significant digits are lost, implementation-defined rounding or truncating occurs, with no exception condition being raised. The rules for arithmetic are specified in Subclause 6.26, "<numeric value expression>".

4.5.2 Operations involving numbers

1. Rationale: Generalise the data type of the return value of an extract expression.

Replace the 3rd bullet of the 1st paragraph with:

— <extract expression> (see Subclause 4.7.3, ''Operations involving datetimes and intervals'') operates on a datetime or interval argument and returns an exact numeric.

4.6.1 Comparison and assignment of booleans

1. Rationale: Clarify assignable and comparable.

Replace the 1st paragraph with:

All boolean values and SQL truth values are comparable and all are assignable to a site of type. The value *true* is greater than the value *false*, and any comparison involving the null value or an *unknown* truth value will return an *unknown* result. The values *true* and *false* may be assigned to any site having a boolean data type; assignment of *unknown*, or the null value, is subject to the nullability characteristic of the target.

4.7.1 Datetimes

1. Rationale: Correct and clarify Table 5.

Insert the following legend to Table 5, "Datetime data type conversions":

SV	Source value
TV	Target value
.UTC	UTC component of *SV* or *TV* (if source or target has time zone)
.TZ	Time zone component of *SV* or *TV* (if source or target has time zone)
STZD	SQL-session default time zone displacement
=>	Cast from the type before the arrow to the type after the arrow
TIME w/ TZ	TIME WITH TIME ZONE
TIME w/o TZ	TIME WITHOUT TIME ZONE
TS w/ TZ	TIMESTAMP WITH TIME ZONE
TS w/o TZ	TIMESTAMP WITHOUT TIME ZONE

In the table cell for the conversion from TIME WITHOUT TIME ZONE to TIMESTAMP WITHOUT TIME ZONE, change "*TZ*" to "*SV*".

In the table cell for the conversion from TIME WITH TIME ZONE to TIMESTAMP WITHOUT TIME ZONE, replace the cell contents with: *SV* => TS w/ TZ => TS w/o TZ.

In the table cell for the conversion from TIMESTAMP WITHOUT TIME ZONE to TIME WITHOUT TIME ZONE, change "*TZ*" to "*SV*"

In the table cell for the conversion from TIMESTAMP WITHOUT TIME ZONE to TIME WITH TIME ZONE, change "TS w/o TZ" to "TIME w/ TZ".

2. *Rationale: Clarify assignable and comparable.*

Replace the 5th paragraph with:

Items of type datetime are comparable only if they have the same <primary datetime field>s.

Insert the following paragraph after the 11th paragraph:

A datetime is assignable to a site only if the source and target of the assignment are both of type DATE, or both of type TIME (regardless whether WITH TIME ZONE or WITHOUT TIME ZONE is specified or implicit), or both of type TIMESTAMP (regardless whether WITH TIME ZONE or WITHOUT TIME ZONE is specified or implicit).

4.7.2 Intervals

1. *Rationale: Clarify assignable and comparable.*

Replace the 8th and 9th paragraphs with:

Year-month intervals are comparable only with other year-month intervals. If two year-month intervals have different interval precisions, they are, for the purpose of any operations between them, effectively converted to the same precision by appending new <primary datetime field>s to either the most significant end of one interval, the least significant end of one interval, or both. New least significant <primary datetime field>s are assigned a value of 0 (zero). When it is necessary to add new most significant datetime fields, the associated value is effectively converted to the new precision in a manner obeying the natural rules for dates and times associated with the Gregorian calendar.

Day-time intervals are comparable only with other day-time intervals. If two day-time intervals have different interval precisions, they are, for the purpose of any operations between them, effectively converted to the same precision by appending new <primary datetime field>s to either the most significant end of one interval or the least significant end of one interval, or both. New least significant <primary datetime field>s are assigned a value of 0 (zero). When it is necessary to add new most significant datetime fields, the associated value is effectively converted to the new precision in a manner obeying the natural rules for dates and times associated with the Gregorian calendar.

4.7.3 Operations involving datetimes and intervals

1. *Rationale: Clarify assignable and comparable.*

Replaced the 3rd paragraph with:

Operations involving values of type datetime require that the datetime values be comparable. Operations involving values of type interval require that the interval values be comparable.

2. *Rationale: Clarify the meaning of the <overlaps predicate>.*

Replace the 5th paragraph with:

<overlaps predicate> uses the operator OVERLAPS to determine whether or not two chronological periods overlap in time. A chronological period is specified either as a pair of datetimes (starting and ending) or as a starting datetime and an interval. If the length of the period is greater than 0, then the period consists of all

points of time greater than or equal to the lower endpoint, and less than the upper endpoint. If the length of the period is 0, the period consists of a single point in time, the lower endpoint. Two periods overlap if they have at least one point in common.

3. *Rationale: Editorial - typographical error.*

Replace the 7th paragraph with:

<interval absolute value function> operates on an interval argument and returns its absolute value in the same most specific type.

4.8 User-defined types

1. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values and correct incorrect terminology and <method characteristic> should not include <transform group specification>. Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace the 3rd paragraph with:

The definition of a user-defined type may include a <method specification list> consisting of one or more <method specification>s. A <method specification> is either an <original method specification> or an <overriding method specification>. Each <original method specification> specifies the <method name>, the <specific method name>, the <SQL parameter declaration list>, the <returns data type>, the <result cast from type> (if any), whether the method is type-preserving, the <language clause>, the <parameter style> if the language is not SQL, whether STATIC or CONSTRUCTOR is specified, whether the method is deterministic, whether the method possibly modifies SQL-data, possibly reads SQL-data, possibly contains SQL, or does not possibly contain SQL, and whether the method should be evaluated as the null value whenever any argument is the null value, without actually invoking the method.

2. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace the 4th paragraph with:

Each <overriding method specification> specifies the <method name>, the <specific method name>, the <SQL parameter declaration list> and the <returns data type>. For each <overriding method specification>, there must be an <original method specification> with the same <method name> and <SQL parameter declaration list> in some proper supertype of the user-defined type. Every SQL-invoked method in a schema must correspond to exactly one <original method specification> or <overriding method specification> associated with some user-defined type existing in that schema.

3. *Rationale: Correct the specification of implicit generation of cast functions when reference values are generated via <user-defined representation>.*

In the 8th paragraph, replace the 12th bullet with:

— If the user-defined type is a structured type, then whether the referencing type of the structured type has a user-defined representation, a derived representation, or a system-defined representation, and the type descriptor of the representation type of the referencing type of the structured type if user-defined representation is specified or the list of attributes if derived representation is specified.

NOTE 5 — "user-defined representation", "derived representation", and "system-defined representation" of a reference type are defined in Subclause 4.10, "Reference types".

4. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace the 1st sub-bullet in the 13th bullet of the 8th paragraph with:

- The <method name>.
- The <specific method name>.

5. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace the 8th sub-bullet in the 13th bullet of the 8th paragraph with:

- If the <method specification> is an <original method specification>, then an indication of whether STATIC or CONSTRUCTOR is specified.

4.8.1 Observers and mutators

1. *Rationale: Attribute values might be null. Clarify the effect of observers and mutuators.*

Replace the 3rd paragraph with:

Let *V* be a value in data type *T* and let *AV* be a value in data type *AT*. The invocation *A*(*V*, *AV*) returns *MV* such that *A*(*MV*) is identical to from *AV* and for every attribute *A'*(*A'* ≠ *A*)of *T*, *A'*(*MV*) is identical to *A'*(*V*). The most specific type of *MV* is the most specific type of *V*.

4.8.2 Constructors

1. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace the 1st and 2nd paragraphs with:

Associated with each structured type *ST* is one *constructor function*, implicitly defined when *ST* is defined. The constructor function is defined if and only if *ST* is instantiable.

Let *TN* be the name of a structured type *T*. The signature of the constructor function for *T* is *TN*( ) and its result data type is *T*. The invocation *TN*( ) returns a value *V* such that *V* is not null and, for every attribute *A* of *T*, *A*(*V*) returns the default value of *A*. The most specific type of *V* is *T*.

For every structured type *ST* that is instantiable, zero or more SQL-invoked constructor methods can be specified. The name of those methods must be equivalent to the name of the type for which they are specified.
NOTE 6.1 — SQL-invoked constructor methods are original methods that cannot be overloaded. An SQL-invoked constructor method and a regular function may exist such that both have equivalent function names, the types of the first parameters (if any) of the method's augmented parameter list and the function's parameter list are the same, and the types of the corresponding remaining parameters (if any) are identical according to the Syntax Rules of Subclause 10.14, "Data type identity".

4.8.4 User-defined type comparison and assignment

1. *Rationale: Values do not have declared types. Every user-defined type has a comparison type. Comparison forms and categories do not have to be the same throughout a subtype family. Correct the restriction on comparison functions*

Replace the 6th paragraph with:

Two values *V1* and *V2* whose most specific types are user-defined types *T1* and *T2* are comparable if and only if *T1* and *T2* are in the same subtype family and each have some comparison type, *CT1* and *CT2* respectively. *CT1* and *CT2* constrain the comparison forms and comparison categories of *T1* and *T2* to be the same. Their respective comparison types *CT1* and *CT2* constrain the comparison forms and comparison categories of *T1* and *T2* to be the same, and the same as those of all their supertypes. If the comparison category is either STATE or RELATIVE, then *T1* and *T2* are constrained to have the same comparison function; if the comparison category is MAP, they are not constrained to have the same comparison function.

4.9 Row types

1. *Rationale: Clarify assignable and comparable.*

Insert the following paragraphs after the 3rd paragraph:

A value of row type *RT1* is assignable to a site of row type *RT2* if and only if the degree of *RT1* is the same as the degree of *RT2* and every field in *RT1* is assignable to the field in the same ordinal position in *RT2*.

A value of row type *RT1* is comparable with a value of row type *RT2* if and only if the degree of *RT1* is the same as the degree of *RT2* and every field in *RT1* is comparable with the field in the same ordinal position in *RT2*.

4.10 Reference types

1. *Rationale: Correct the specification of implicit generation of cast functions when reference values are generated via <user-defined representation>.*

Replace the 2nd paragraph with:

Given a structured type *T*, the REF values that can reference rows in typed tables defined on *T* collectively form a certain data type *RT* known as a *reference type*. *RT* is the *referencing type* of *T*. *T* is the referenced type of *RT*.

2. *Rationale: Add the notion of a type designator for reference types.*

Insert the following paragraph after the 2nd paragraph:

Let *TN* be the name of *T*. The type designator of *RT* is REF(*TN*).

3. *Rationale: Clarify assignable and comparable.*

Replace the 4th paragraph with:

Values of two reference types are comparable if the referenced types of their declared types have some common supertype.

4. *Rationale: Correct definition of the length of <reference type>s and add the notion of a type designator for reference types.*

Replace the 7th paragraph and its associated bullet items with:

A reference type is described by a reference type descriptor. The reference type descriptor for *RT* includes:

— The type designator of *RT*.

— The name of the referenceable table, if any, that is the scope of the reference type.

5. *Rationale: Correct definition of the length of <reference type>s.*

Replace the 8th paragraph with:

In a host variable, a REF value is materialized as an *N*-octet value, where *N* is the length of the <reference type>.

4.11 Collection types

1. *Rationale: Add the notion of a type designator for collection types.*

Insert the following paragraph after the 2nd paragraph:

Let *EDTN* be the type designator of *EDT*. The type designator of *CT* is *EDTN* ARRAY.

2. *Rationale: Add the notion of a type designator for collection types.*

Replace the 3rd paragraph with:

A *collection type descriptor* describes a collection type. The collection type descriptor for *CT* includes:

— The type designator of *CT*.

— The descriptor of the element type of *CT*.

— An indication that *CT* is an array type.

4.11.2 Collection comparison

1. Rationale: Clarify assignable and comparable.

Replace the 1st paragraph with:

> A value of collection type *CT1* is assignable to a site of collection type *CT2* if and only if the element type of *CT1* is assignable to the element type of *CT2*.

4.12 Type conversions and mixing of data types

1. Rationale: Clarify assignable and comparable.

Delete the entire subclause.

4.13 Data conversions

1. Rationale: Clarify assignable and comparable.

Insert the following paragraph before the 1st paragraph:

> Implicit type conversion can occur in expressions, fetch operations, single row select operations, inserts, deletes, and updates. Explicit type conversions can be specified by the use of the CAST operator.

2. Rationale: To clarify the concept of user-defined casts.

Replace the 1st paragraph with:

> Explicit data conversions can be specified by a *CAST operator*. A CAST operator defines how values of a source data type are converted into a value of a target data type according to the Syntax Rules and General Rules of Subclause 6.22, "<cast specification>". Data conversions between predefined data types and between constructed types are defined by the rules of this part of ISO/IEC 9075. Data conversions between a user-defined type and another data type are defined by a user-defined cast.

4.16 Tables

1. Rationale: Clean up typed table insertability property for non-instantiable types and define the term updatable column for base tables.

Replace the 7th paragraph with:

> All base tables are *updatable*. Every column of a base table is an *updatable column*. Derived tables are either updatable or not updatable. The operations of update and delete are permitted for updatable tables, subject to constraining Access Rules. Some updatable tables, including all base tables whose row type is not derived from a user-defined type that is not instantiable, are also *insertable-into*, in which case the operation of insert is also permitted, again subject to Access Rules.

2. *Rationale: Clean up typed table insertability property for non-instantiable types.*

Replace the 11th paragraph with:

Every table descriptor includes:

— The column descriptor of each column in the table.

— The name, if any, of the structured type, if any, associated with the table.

— An indication of whether the table is insertable-into or not.

— An indication of whether the base table is a referenceable table or not, and an indication of whether the self-referencing column is a system-generated, a user-generated, or a derived self-referencing column.

— A list, possibly empty, of the names of its direct supertables.

— A list, possibly empty, of the names of its direct subtables.

3. *Rationale: Correct the default of <table commit action>.*

Replace the 3rd bullet of 12th paragraph with:

— If the base table is a global temporary table, a created local temporary table, or a declared local temporary table, then an indication of whether ON COMMIT PRESERVE ROWS was specified or ON COMMIT DELETE ROWS was specified or implied.

4. *Rationale: Clean up typed table insertability property for non-instantiable types.*

Replace the 13th paragraph with:

A derived table descriptor describes a derived table. In addition to the components of every table descriptor, a derived table descriptor includes:

— The <query expression> that defines how the table is to be derived.

— An indication of whether the derived table is updatable or not.

4.16.1 Types of tables

1. *Rationale: Correct the definition of group of a grouped table to use non-distinctness rather than equality.*

Replace the 7th paragraph with:

A *grouped table* is a set of groups derived during the evaluation of a <group by clause>. A group *G* is a collection of rows in which, for every grouping column *GC*, if the value of *GC* in some row is *GV*, then the value of *GC* in every row is not distinct from *GV*; moreover, if *R1* is a row in group *G1* of grouped table *GT* and *R2* is a row in *GT* such that for every grouping column *GC* the value of *GC* in *R1* is not distinct from the value of *GC* in *R2*, then *R2* is in *G1*. Every row in *GT* is in exactly one group. A group may be considered as a table. Set functions operate on groups.

4.16.3 Operations involving tables

1. Rationale: Correct the description of the effect of deleting, updating and inserting a row in a table.

Replace the 4th, 5th and 6th paragraphs with:

Let *T* be a table whose value is *TV1*. For every row *RR*, let n_{RR} be the number of rows in *TV1* that are identical to *RR*.

The effect of deleting a row *R* from *T* is to replace the value *TV1* of *T* with the value *TV2* such that, for *RR* identical to *R*, the number of rows in *TV2* that are identical to *RR* is n_{RR} - 1 (one) and for *RR* not identical to *R*, the number of rows in *TV2* that are not identical to *RR* is n_{RR}. The primary effect of a <delete statement: positioned> on *T* is to delete exactly one specified row from *T*. The primary effect of a <delete statement: searched> on *T* is to delete zero or more rows from *T*.

The effect of replacing a row *R1* with the row *R2* in *T* is to replace the value *TV1* of *T* with the value *TV2* such that, if *R1* is not identical to *R2*, then, for *RR* identical to *R1*, the number of rows in *TV2* that are identical to *RR* is n_{RR} - 1 (one), for *RR* identical to *R2*, the number of rows in *TV2* that are identical to *RR* is n_{RR} + 1 (one), and for *RR* identical to neither *R1* nor *R2*, the number of rows in *TV2* that are identical to *RR* is n_{RR}. Otherwise *R1* is identical to *R2* and the number of rows in *TV2* that are identical to *RR* is n_{RR} for all *RR*.

The primary effect of an <update statement: positioned> on *T* is to replace exactly one specified row in *T* with some specified row. The primary effect of an <update statement: searched> on *T* is to replace zero or more rows in *T*.

If *T*, as well as being updatable, is insertable-into, then rows can be inserted into it. The effect of inserting a row *R* into *T* is to replace the value *TV1* of *T* with the value *TV2* such that, for *RR* identical to *R*, the number of rows in *TV2* that are identical to *RR* is n_{RR} + 1 (one) and, for *RR* not identical to *R*, the number of rows in *TV2* that are identical to *RR* is n_{RR}. The primary effect of an <insert statement> on *T* is to insert into *T* each of the zero or more rows contained in a specified table.

2. Rationale: Correct reference to syntactic construct.

Replace 2nd bullet of the 8th paragraph with:

— Updatable views whose <view definition>s do not specify WITH CASCADED CHECK OPTION, which might result in alteration or reversal of primary effects.

4.17 Integrity constraints

1. Rationale: state the correct limitation on integrity constraints regarding determinism.

Replace the 3rd paragraph with:

No integrity constraint shall be defined using a <search condition> that generally contains a possibly non-deterministic <value expression>.

4.17.1 Checking of constraints

1. Rationale: Address requirement for multiple diagnostics areas

Replace the 4th paragraph with:

> When a constraint is checked other than at the end of an SQL-transaction, if it is not satisfied, then an exception condition is raised and the SQL-statement that caused the constraint to be checked has no effect other than entering the exception information into the first diagnostics area. When a <commit statement> is executed, all constraints are effectively checked and, if any constraint is not satisfied, then an exception condition is raised and the SQL-transaction is terminated by an implicit <rollback statement>.

4.17.2 Table constraints

1. Rationale: Replace reference to non-existent BNF non-terminal with the correct non-terminal.

Replace Note 14 with:

> NOTE 14 – If MATCH FULL or MATCH PARTIAL is specified for a referential constraint and if the referencing table has only one column specified in <referential constraint definition> for that referential constraint, or if the referencing table has more than one specified column for that <referential constraint definition>, but none of those columns is nullable, then the effect is the same as if no <match type> were specified.

Replace the 9th paragraph with:

> A referential constraint is satisfied if one of the following conditions is true, depending on the <match type> specified in the <referential constraint definition>:
>
> — If no <match type> was specified then, for each row *R1* of the *referencing table*, either at least one of the values of the *referencing columns* in *R1* shall be a null value, or the value of each referencing column in *R1* shall be equal to the value of the corresponding *referenced column* in some row of the *referenced table*.
>
> — If MATCH FULL was specified then, for each row *R1* of the *referencing table*, either the value of every *referencing column* in *R1* shall be a null value, or the value of every *referencing column* in *R1* shall not be null and there shall be some row *R2* of the *referenced table* such that the value of each *referencing column* in *R1* is equal to the value of the corresponding *referenced column* in *R2*.
>
> — If MATCH PARTIAL was specified then, for each row *R1* of the *referencing table*, there shall be some row *R2* of the *referenced table* such that the value of each *referencing column* in *R1* is either null or is equal to the value of the corresponding *referenced column* in *R2*.

4.18.1 General rules and definitions

1. Rationale: Use the notion of distinct.

Replace the 2nd paragraph with:

Let "*T*: *A* ↬ *B*" (read "in *T*, *A* determines *B*" or "*B* is functionally dependent on *A* in *T*") denote the functional dependency of *B* on *A* in *T*, which is true if, for any possible value of *T*, any two rows that are not distinct for every column in *A* also are not distinct for every column in *B*. When the table *T* is understood from context, the abbreviation "*A* ↬ *B*" may also be used.

2. Rationale: Correct erroneous symbol.

Replace the 8th paragraph with:

In the following Subclauses, let a column *C1* be a *counterpart* of a column *C2* under qualifying table *QT* if *C1* is specified by a column reference (or by a <value expression> that is a column reference) that references *C2* and *QT* is the qualifying table of *C2*. If *C1* is a counterpart of *C2* under qualifying table *QT1* and *C2* is a counterpart of *C3* under qualifying table *QT2*, then *C1* is a counterpart of *C3* under *QT2*.

3. Rationale: Editorial.

Replace the 19th paragraph with:

Let *AC* be an AND-component of *SC* such that *AC* is a <comparison predicate> whose <comp op> is <equals operator>. Let *RVE1* and *RVE2* be the two <row value expression>s that are the operands of *AC*. Suppose that both *RVE1* and *RVE2* are <row value constructor>s. Let *n* be the number of <row value constructor element>s in *RVE1*. Let $RVEC1_i$ and $RVEC2_i$, 1 (one) ≤ *i* ≤ *n*, be the *i*-th <row value constructor element> of *RVE1* and *RVE2*, respectively. The <comparison predicate> "$RVEC1_i = RVEC2_i$" is called an *equality AND-component* of *SC*.

4.18.4 Known functional dependencies in a <joined table>

1. Rationale: Editorial.

Replace NOTE 23 with:

NOTE 23 — An SQL-implementation may also choose to recognize similar known functional dependencies of the form { CRA_1, . . . , CRA_N } ↬ {*CRBC*} in case one comparand is a deterministic expression of column references CRA_1, . . . , CRA_N under similar conditions.

4.18.9 Known functional dependencies in the result of <having clause>

1. Rationale: More than one <search condition> may be directly contained in a <having clause>.

Replace the 1st paragraph with:

Let *T1* be the table that is the operand of the <having clause>, let *SC* be the <search condition> simply contained in the <having clause>, and let *R* be the result of the <having clause>.

4.18.10 Known functional dependencies in a <query specification>

1. Rationale: correct the specification of functional dependencies for derived columns.

Replace the 1st paragraph with:

Let *T* be the <table expression> simply contained in the <query specification> *QS* and let *R* be the result of the <query specification>.

Replace the 6th and 7th paragraphs with:

Let *CC* be the column specified by some <value expression> *VE* that is not possibly non-deterministic in the <select list>.

Let *OP1*, *OP2*, ... be the operands of *VE* that are column references whose qualifying query is *QS* and that are not contained in an aggregated argument of a <set function specification>.

If *VE* does not contain a <set function specification> whose aggregation query is *QS*, then {*OP1* , *OP2*, ... } ➞ *CC* is a known functional dependency in *T1*.

If *VE* contains a <set function specification> whose aggregation query is *QS*, then let {*G1* , ...} be the set of grouping columns of *T*. Then {*G1* ,..., *OP1* , *OP2*, ... } ➞ *CC* is a known functional dependency in *T1*.

4.20 SQL-schemas

1. Rationale: Cleanup of imprecise wording.

Replace the 5th paragraph with:

Base tables and views are identified by <table name>s. A <table name> consists of a <schema name> and an <identifier>. The <schema name> identifies the schema in which a persistent base table or view identified by the <table name> is defined. Base tables and views defined in different schemas can have <identifier>s that are equal according to the General Rules of Subclause 8.2, "comparison predicate>".

4.21 SQL-client modules

1. Rationale: Editorial.

Replace NOTE 29 with:

NOTE 29 — The <module character set specification> has no effect on the SQL language contained in the SQL-client module and exists only for compatibility with ISO/IEC 9075:1992. It may be used to document the character set of the SQL-client module.

4.23 SQL-invoked routines

1. *Rationale: Correct the definition of an SQL-invoked routine that is "dependent on a user-defined type" to include mutator and observer functions that are implicitly created by the execution of the <alter type statement>.*

Replace the 3rd paragraph with:

An SQL-invoked routine *SR* is said to be *dependent* on a user-defined type *UDT* if *SR* is created during the execution of the <user-defined type definition> that created *UDT* or if *SR* is created during the execution of the <alter type statement> that specifies an <add attribute definition>. An SQL-invoked routine that is dependent on a user-defined type cannot be modified by an <alter routine statement> nor be destroyed by a <drop routine statement>. It is destroyed implicitly by a <drop data type statement>.

2. *Rationale: Editorial.*

Replace the 4th, 5th and 6th paragraphs with:

A <predicate> *P* is said to be dependent on an SQL-invoked routine *SR* if and only if *SR* is the ordering function included in the descriptor of a user-defined type *UDT* and one of the following conditions is true:

— *P* is a <comparison predicate> that immediately contains a <row value expression> whose declared type is some user-defined type *T1* whose comparison type is *UDT*.

— *P* is a <quantified comparison predicate> that immediately contains a <row value expression> that has some field whose declared type is some user-defined type *T1* whose comparison type is *UDT*.

— *P* is a <unique predicate> that immediately contains a <table subquery> that has a column whose declared type is some user-defined type *T1* whose comparison type is *UDT*.

— *P* is a <match predicate> that immediately contains a <row value expression> that has some field whose declared type is some user-defined type *T1* whose comparison type is *UDT*.

— *P* is a <comparison predicate> with some corresponding value whose declared type is some array type whose element type is a user-defined type *T1* whose comparison type is *UDT*.

— *P* is a <quantified comparison predicate> that immediately contains a <row value expression> that has some field whose declared type is some array type whose element type is a user-defined type *T1* whose comparison type is *UDT*.

— *P* is a <unique predicate> that immediately contains a <table subquery> that has a column whose declared type is some array type whose element type is a user-defined type *T1* whose comparison type is *UDT*.

— *P* is a <match predicate> that immediately contains a <row value expression> that has some field whose declared type is some array type whose element type is a user-defined type *T1* whose comparison type is *UDT*.

NOTE 30 — "Comparison type" is defined in Subclause 4.8.4, "User-defined type comparison and assignment".

A <set function specification> *SFS* is said to be dependent on an SQL-invoked routine *SR* if and only if all the following are true:

— *SR* is the ordering function included in the descriptor of a user-defined type *UDT*.

— *SFS* is a <general set function> whose <set function type> is MAX or MIN or *SFS* is a <general set function> whose <set qualifier> is DISTINCT.

— The declared type of the <value expression> of *SFS* is *UDT*.

A <group by clause> *GBC* is said to be dependent on an SQL-invoked routine *SR* if and only if all the following are true:

— *SR* is the ordering function included in the descriptor of a user-defined type *UDT*.

— The declared type of a grouping column of *GBC* is *UDT*.

3. *Rationale: Correct the description of type-preserving functions.*

Replace the 8th paragraph with:

An *SQL-invoked function* is an SQL-invoked routine whose invocation returns a value. Every parameter of an SQL-invoked function is an input parameter, one of which may be designated as the result SQL parameter. The format of an SQL-invoked function is specified by <SQL-invoked function> (see Subclause 11.49, "<SQL-invoked routine>"). An SQL-invoked function can be a *type-preserving function*; a type-preserving function is an SQL-invoked function that has a result SQL parameter. The most specific type of a non-null result of invoking a type-preserving function must be compatible with the most specific type of the value of the argument substituted for its result SQL parameter.

4. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace the 9th paragraph that begins with "An SQL-invoked method is an SQL-invoked function ..." with:

An *SQL-invoked method* is an SQL-invoked function that is specified by <method specification designator> (see Subclause 11.49, "<SQL-invoked routine>"). There are three kinds of SQL-invoked methods: *instance SQL-invoked methods*, *SQL-invoked constructor methods*, and *static SQL-invoked methods*. All SQL-invoked methods are associated with a user-defined type, also known as the *type of the method*. The <method characteristics> of an SQL-invoked method are specified by a <method specification> contained in the <user-defined type definition> of the type of the method. An instance SQL-invoked method and an SQL-invoked constructor method both satisfy the following conditions:

— Its first parameter, called the *subject parameter*, has a declared type that is a user-defined type. The type of the subject parameter is the type of the method. A parameter other than the subject parameter is called an *additional parameter*.

— Its descriptor is in the same schema as the descriptor of the data type of its subject parameter.

An SQL-invoked constructor method satisfies the following additional condition:

— Its <method name> is equivalent to the <qualified identifier> simply contained in the <user-defined type name> included in the user-defined type descriptor of the type of the method.

5. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace the 16th paragraph which begins with "SQL-invoked routines are invoked differently…" with:

SQL-invoked routines are invoked differently depending on their form. SQL-invoked procedures are invoked by <call statement>s. SQL-invoked regular functions are invoked by <routine invocation>s. Instance SQL-invoked methods are invoked by <method invocation>s, while SQL-invoked constructor methods are invoked by <new invocation>s and static SQL-invoked methods are invoked by <static method invocation>s. An invocation of an SQL-invoked routine specifies the <routine name> of the SQL-invoked routine and supplies a sequence of argument values corresponding to the <SQL parameter declaration>s of the SQL-invoked routine. A *subject routine* of an invocation is an SQL-invoked routine that may be invoked by a <routine invocation>. After the selection of the subject routine of a <routine invocation>, the SQL arguments are evaluated and the SQL-invoked routine that will be executed is selected. If the subject routine is an instance SQL-invoked method, then the SQL-invoked routine that is executed is selected from the set of overriding methods of the subject routine. (The term "set of overriding methods" is defined in the General Rules of Subclause 10.4, "<routine invocation>".) The overriding method that is selected is the overriding method with a subject parameter the type designator of whose declared type precedes that of the declared type of the subject parameter of every other overriding method in the type precedence list of the most specific type of the value of the SQL argument that corresponds to the subject parameter. See the General Rules of Subclause 10.4, "<routine invocation>". If the subject routine is not an SQL-invoked method, then the SQL-invoked routine executed is that subject routine. After the selection of the SQL-invoked routine for execution, the values of the SQL arguments are assigned to the corresponding SQL parameters of the SQL-invoked routine and its <routine body> is executed. If the SQL-invoked routine is an SQL routine, then the <routine body> is an <SQL procedure statement> that is executed according to the General Rules of <SQL procedure statement>. If the SQL-invoked routine is an external routine, then the <routine body> identifies a program written in some standard programming language other than SQL that is executed according to the rules of that standard programming language.

6. *Rationale: Editorial.*

Replace the 20th paragraph, which begins "If a <routine invocation> is contained in a <query expression> of a view, a check constraint, or an assertion, …", with:

If a <routine invocation> is contained in a <query expression> of a view, a check constraint, or an assertion, the <triggered action> of a trigger, or in an <SQL-invoked routine>, then the subject routine for that invocation is determined at the time the view is created, the check constraint is defined, the assertion is created, the trigger is created, or the SQL-invoked routine is created. If the subject routine is an SQL-invoked procedure, an SQL-invoked regular function, or a static SQL-invoked method, then the same SQL-invoked routine is executed whenever the view is used, the check constraint or assertion is evaluated, the trigger is executed, or the SQL-invoked routine is invoked. If the subject routine is an instance SQL-invoked method, then the SQL-invoked routine that is executed is determined whenever the view is used, the check constraint or assertion is evaluated, the trigger is executed, or the SQL-invoked routine is invoked, based on the most specific type of the value resulting from the evaluation of the SQL argument that correspond to the subject parameter. See the General Rules of Subclause 10.4, "<routine invocation>".

7. *Rationale: use "identical" instead of "same" or "equal" and remove cut and paste error.*

Replace the 22nd paragraph with:

An SQL-invoked routine is either *deterministic* or *possibly non-deterministic*. An SQL-invoked function that is deterministic always returns the identical return value for a given list of SQL argument values. An SQL-

invoked procedure that is deterministic always returns the identical values in its output and inout SQL parameters for a given list of SQL argument values. An SQL-invoked routine is possibly non-deterministic if it might produce nonidentical results when invoked with the identical list of SQL argument values.

8. *Rationale: Clarify the semantics of SQL-data access indication.*

Replace the 23rd paragraph, which begins "An external routine either *does not possibly contain SQL* or *possibly contains SQL*...", with:

An external routine *does not possibly contain SQL*, *possibly contains SQL*, *possibly reads SQL-data*, or *possibly modifies SQL-data*. Only an external routine that possibly contains SQL, or possibly reads SQL-data or possibly modifies SQL-data may execute SQL-statements during its invocation.

Replace the 24th paragraph, which begins "An SQL-invoked routine may or may not *possibly read SQL-data*.", with:

An SQL routine *possibly contains SQL*, *possibly reads SQL-data*, or *possibly modifies SQL-data*. Only an SQL-invoked routine that possibly reads SQL-data or possibly modifies SQL-data may read SQL-data during its invocation.

Replace the 25th paragraph, which begins "An SQL-invoked routine may or may not *possibly modify SQL-data*.", with:

Only an SQL-invoked routine that *possibly modifies SQL-data* may modify SQL-data during its invocation.

9. *Rationale: Missing word.*

Replace the 28th paragraph, which begins "The identifiers in this new entry of the authorization stack are then modified", with:

The identifiers in this new entry of the authorization stack are then modified depending on whether the SQL-invoked routine is an SQL routine or an external routine. If the SQL-invoked routine is an SQL routine, then, if the routine authorization identifier is a user identifier, the user identifier is set to the routine authorization identifier and the role name is set to null; otherwise, the role name is set to the routine authorization identifier and the user identifier is set to null.

4.24 Built-in functions

1. *Rationale: Clarify the semantics of built-in functions.*

Delete the entire subclause.

4.25 SQL-paths

1. Rationale: Use the correct BNF non-terminal (<user-defined type> rather than <user-defined type name>).

Replace the 2nd bullet of the 1st paragraph with:

— The user-defined type when the <user-defined type> does not contain a <schema name>.

2. Rationale: Clarify the semantics of built-in functions.

Delete the 2nd paragraph.

4.26.4 Locators

1. Rationale: Correct the specification of which locators are marked invalid when an SQL-transaction ends.

Replace 5th paragraph with:

A non-holdable locator remains valid until the end of the SQL-transaction in which it was generated, unless it is explicitly made invalid by the execution of a <free locator statement> or a <rollback statement> that specifies a <savepoint clause> is executed before the end of that SQL-transaction if the locator was generated subsequent to the establishment of the savepoint identified by the <savepoint clause>.

Replace 6th paragraph with:

A holdable locator may remain valid beyond the end of the SQL-transaction in which it is generated. A holdable locator becomes invalid whenever a <free locator statement> identifying that locator is executed or the SQL-transaction in which it is generated or any subsequent SQL-transaction is rolled back. All locators remaining valid at the end of an SQL-session are marked invalid when that SQL-session terminates.

4.27 Diagnostics area

1. Rationale: Address requirement for multiple diagnostics areas and rationalisation of terminology.

Replace the entire subclause with:

A diagnostics area is a place where completion and exception condition information is stored when an SQL-statement is executed. The diagnostics areas associated with an SQL-agent form *the diagnostics area stack* of that SQL-agent. For definitional purposes, the diagnostics areas in this stack are considered to be numbered. There is one diagnostics area stack associated with an SQL-agent, regardless of the number of SQL-client modules that the SQL-agent includes or the number of connections in use.

Two operations on diagnostics area stacks are specified in this international standard for definitional purposes only. *Pushing* a diagnostics area stack effectively creates a new first diagnostics area, incrementing the ordinal position of every existing diagnostics area in the stack by 1 (one). The content of the new first diagnostics area is initially a copy of the content of the old, now second, one. *Popping* a diagnostics area stack effectively destroys the first diagnostics area in the stack and decrements the ordinal position of every diagnostics area by 1 (one). The maximum number of diagnostics areas in a diagnostics area stack is implementation-dependent.

A diagnostics area consists of a *statement area* and a sequence of one or more *condition areas*, each of which is at any particular time either *occupied* or *vacant*. A diagnostics area is *empty* when each of its condition areas is vacant; *emptying* a diagnostics area brings about this state.

A statement area consists of a collection of named *statement information items*. A condition area consists of a collection of named *condition information items*.

At the beginning of the execution of any <SQL procedure statement> that is not an <SQL diagnostics statement>, the first diagnostics area is emptied. An implementation places information about a completion condition or an exception condition reported via SQLSTATE into a vacant condition area in this diagnostics area. If other conditions are raised, the extent to which these cause further condition areas to become occupied is implementation-defined.

An <externally-invoked procedure> containing an <SQL diagnostics statement> returns a code indicating a completion or exception condition for that statement via SQLSTATE, but does not necessarily cause any vacant condition areas to become occupied.

The number of condition areas per diagnostics area is referred to as the *diagnostics area limit*. An SQL-agent may set the diagnostics area limit with the <set transaction statement>; if the SQL-agent does not specify the diagnostics area limit, then the diagnostics area limit is implementation-dependent, but shall be at least one. An SQL-implementation may place information into this area about fewer conditions than there are condition areas. The ordering of the information about conditions placed into a diagnostics area is implementation-dependent, except that the first condition area in a diagnostics area always corresponds to the condition specified by the SQLSTATE value.

The <get diagnostics statement> is used to obtain information from an occupied condition area, referenced by its ordinal position within the first diagnostics area.

4.30.1 Classes of SQL-statements

1. Rationale: Clarify the semantics of SQL-data access indication.

Replace the 2nd paragraph with:

In this part of ISO/IEC 9075, there are at least three ways of classifying SQL-statements:

— According to their effect on SQL objects, whether persistent objects, i.e., SQL-data, SQL-client modules, and schemas, or transient objects, such as SQL-sessions and other SQL-statements.

— According to whether or not they start an SQL-transaction, or can, or must, be executed when no SQL-transaction is active.

— According to whether they possibly read SQL-data or possibly modify SQL-data.

4.30.2 SQL-statements classified by functions

1. Rationale: Correct the classification of SQL-statements.

Replace the 13th bullet of the 2nd paragraph with:

— <grant privilege statement>

2. Rationale: Correct the classification of SQL-statements.

Insert the following bullets to the 2nd paragraph:

— <user-defined cast definition>

— <drop user-defined cast statement>

3. Rationale: Clarify the semantics of SQL-data access indication.

Insert the following Subclause after Subclause 4.30.2, "SQL-statements classified by functions":

4.30.2a SQL-statements and SQL-data access indication

The following SQL-statements possibly read SQL-data:

— SQL-data statements other than SQL-data change statements, <free locator statement>, or and <hold locator statement>

— SQL-statements that simply contain a <subquery>

The following SQL-statements possibly modify SQL-data:

— SQL-schema statements

— SQL-data change statements

NOTE 31.1 — SQL-transaction statements, SQL-connection statements and SQL-dynamic statements are not included in the above classification since they are not permitted to occur in SQL-invoked routines.

4.30.3 SQL-statements and transaction states

1. Rationale: Correct the classification of SQL-statements.

Replace the 3rd paragraph with:

The following SQL-statements are possibly transaction-initiating SQL-statements:

— <return statement>

— <call statement>

4.30.4 SQL-statement atomicity

1. Rationale: Add missing concept and clarify that not all SQL-statements are atomic.

Replace the entire Subclause with:

The execution of all SQL-statements other than certain SQL-control statements is atomic with respect to recovery. Such an SQL-statement is called an *atomic SQL-statement*.

An SQL-statement that is not an atomic SQL-statement is called a *non-atomic SQL-statement*.

The following are non-atomic SQL-statements:

— <call statement>

— <commit statement>

— <execute statement>

— <execute immediate statement>

— <return statement>

— <rollback statement>

— <savepoint statement>

All other SQL-statements are atomic.

A statement execution context is either *atomic* or *non-atomic*.

The execution of a non-atomic SQL-statement causes a non-atomic execution context to exist.

The execution of an atomic SQL-statement or evaluation of a <subquery> causes an atomic execution context to exist.

Within one execution context, another execution context may become active. This latter execution context is said to be a *more recent execution context* than the former. If there is no execution context that is more recent than execution context *EC*, then *EC* is said to be the *most recent execution context*.

If there is no atomic execution context that is more recent than atomic execution context *AEC*, then *AEC* is the *most recent atomic execution context*.

An SQL-transaction cannot be explicitly terminated within an atomic execution context. If the execution of an atomic SQL-statement is unsuccessful, then the changes to SQL-data or schemas made by the SQL-statement are cancelled.

4.31.1 Authorization identifiers

1. Rationale: Editorial.

Replace the 1st paragraph that begins with "An <authorization identifier> identifies a set of privileges. ..." with:

> An <authorization identifier> identifies a set of privileges. An <authorization identifier> can be either a <user identifier> or a <role name>. A <user identifier> represents a user of the database system. The mapping of <user identifier>s to operating system users is implementation-dependent.
> A <role name> represents a role.

4.32 SQL-transactions

1. Rationale: Address requirement for multiple diagnostics areas

Replace the 5th paragraph with:

> It is implementation-defined whether or not, or how, a <rollback statement> that references a <savepoint specifier> affects diagnostics area contents, the contents of SQL descriptor areas, and the status of prepared statements.

2. Rationale: Address requirement for multiple diagnostics areas

Replace the 8th paragraph with:

> An SQL-transaction has a *diagnostics area limit* which is a positive integer that specifies the maximum number of conditions that can be placed in any diagnostics area during execution of an SQL-statement in this SQL-transaction.

3. Rationale: Closing a cursor does not cause any effective check of constraints, etc.

Delete the 9th paragraph.

4. Rationale: Correct reference to incorrect attribute (access mode instead of isolation level).

Replace the 11th paragraph with:

> An SQL-transaction has an isolation level that is READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE. The isolation level of an SQL-transaction defines the degree to which the operations on SQL-data or schemas in that SQL-transaction are affected by the effects of and can affect operations on SQL-data or schemas in concurrent SQL-transactions. The isolation level of an SQL-transaction when any cursor is held open from the previous SQL-transaction within an SQL-session is the isolation level of the previous SQL-transaction by default. If no cursor is held open, or this is the first SQL-transaction within an SQL-session, then the isolation level is SERIALIZABLE by default. The level can be explicitly set by the <set transaction statement> before the start of an SQL-transaction or by the use of a <start transaction statement> to start an SQL-transaction. If it is not explicitly set, then the isolation level is implicitly set to the default isolation level for the SQL-session before each SQL-transaction begins. If no <set session characteristics statement> has set the default isolation level for the SQL-session, then the default isolation level for the SQL-session is SERIALIZABLE.

4.34 SQL-sessions

1. Rationale: Add definition of executing statement.

Replace the 1st paragraph with:

An *SQL-session* spans the execution of a sequence of consecutive SQL-statements invoked by a single user from a single SQL-agent. At any one time during an SQL-session, exactly one of the SQL-statements in this sequence is being executed and is said to be an executing statement. In some cases, an executing statement *ES* causes a nested sequence of consecutive SQL-statements to be executed as a direct result of *ES*; during that time exactly one of these is also an executing statement and it in turn might similarly involve execution of a further nested sequence, and so on, indefinitely. An executing statement *ES* such that no statement is executing as a direct result of *ES* is called the innermost executing statement of the SQL-session.

2. Rationale: Address requirement for multiple diagnostics areas and rationalisation of terminology

In the 8th paragraph replace the 11th bullet with:

— The current SQL diagnostics area and its contents, along with the current diagnostics area limit.

4.34.1 Execution contexts

1. Rationale: Correct cross reference and clarify that not all SQL-statements are atomic.

Replace the 1st paragraph with:

Execution contexts augment an SQL-session context to cater for certain special circumstances that might pertain from time to time during invocations of SQL-statements. An execution context is either a statement execution context, a trigger execution context or a routine execution context. There is always a statement execution context, a routine execution context, and zero or more trigger execution contexts. For certain SQL-statements, the statement execution context is always atomic; for others, it is always or sometimes non-atomic. A routine execution context is either atomic or non-atomic. Every trigger execution context is atomic. Statement execution contexts are described in Subclause 4.30.4, "SQL-statement atomicity". Trigger execution contexts are described in Subclause 4.35, "Triggers".

2. Rationale: Clarify the semantics of SQL-data access indication.

Replace the 2nd paragraph with:

A routine execution context consists of:

— An indication as to whether or not an SQL-invoked routine is active.

— An SQL-data access indication, which indicates what SQL-statements, if any, are allowed during the execution of an SQL-invoked routine. The SQL-data access indication is one of the following: does not possibly contain SQL, possibly contains SQL, possibly reads SQL-data, or possibly modifies SQL-data.

— An identification of the SQL-invoked routine that is active.

— The routine SQL-path derived from the routine SQL-path if the SQL-invoked routine that is active is an SQL routine and from the external routine SQL-path if the SQL-invoked routine that is active is an external routine.

4.35 Triggers

1. Rationale: Editorial.

Replace the 2nd paragraph with:

A *BEFORE trigger* is a trigger whose trigger action time specifies BEFORE. An *AFTER trigger* is a trigger whose trigger action time specifies AFTER.

4.35.2 Execution of triggers

1. Rationale: Editorial.

Replace the 2nd paragraph that begins with "A trigger execution context consists of a set of ..." with:

A trigger execution context consists of a set of *state changes*. Within a trigger execution context, each state change is uniquely identified by a *trigger event*, a *subject table*, and a *column list*. The trigger event can be DELETE, INSERT, or UPDATE. A state change *SC* contains a *set of transitions*, a set of statement-level triggers *considered as executed* for *SC*, and a set of row-level triggers, each paired with the set of rows in *SC* for which it is considered as executed.

2. Rationale: Clarify the notion of state changes.

Replace the 9th paragraph with:

When execution of an SQL-data change statement S_i causes a trigger execution context TEC_i to come into existence, the set of state changes SSC_i is empty. Let *TE* be one of DELETE, INSERT, or UPDATE contained in S_i. Let *ST* be the subject table of S_i. If *TE* is INSERT or DELETE, then let *PSC* be a set whose only element is the empty set. If *TE* is UPDATE, then let *CL* be the list of columns being updated by S_i, and let *OC* be the set of column names identifying the columns in *CL*. Let *PSC* be the set consisting of the empty set and every subset of the set of column names of *ST* that has at least one column name that is in *OC*. Let *PSCN* be the number of elements in *PSC*. A state change SC_{ij} for *j* varying from 1 (one) to *PSCN*, identified by *TE*, *ST*, and the *j*-th element in *PSC*, is added to SSC_i, provided SSC_i does not already contain a state change corresponding to SC_{ij}. A transition T_{ijk} is added to SC_{ij} when a row is inserted into, updated in, or deleted from *ST* during the execution of S_i or the checking of referential constraints according to the General Rules of Subclause 11.8, "<referential constraint definition>", Subclause 14.6, "<delete statement: positioned>", Subclause 14.7, "<delete statement: searched>", Subclause 14.8, "<insert statement>", Subclause 14.9, "<update statement: positioned>", and Subclause 14.10, "<update statement: searched>".

3. Rationale: Correct the use of unidentified identifiers.

Replace the 12th paragraph with:

When a state change SC_{ij} arises in SSC_i, one or more triggers are activated by SC_{ij}. A trigger *TR* is activated by SC_{ij} if and only if the subject table of *TR* is the subject table of SC_{ij}, the trigger event of *TR* is the trigger event of SC_{ij}, and the set of column names listed in the trigger column list of *TR* is the equivalent to the set of column names of SC_{ij}.

NOTE 33.1 — The trigger column list is included in the descriptor of *TR*; it is empty if the trigger event is DELETE or INSERT. The trigger column list is also empty if the trigger event is UPDATE, but the <trigger event> of the <trigger definition> that defined *TR* does not specify a <trigger column list>.

4.36 Client-server operation

1) *Rationale: Address requirement for multiple diagnostics areas*

Replace the 4th paragraph with:

A call by the SQL-agent to an <externally-invoked procedure> whose <SQL procedure statement> simply contains an <SQL diagnostics statement> fetches information from the specified diagnostics area in the diagnostics area stack associated with the SQL-client. Following the execution of an <SQL procedure statement> by an SQL-server, diagnostic information is passed in an implementation-dependent manner into the SQL-agent's diagnostics area stack in the SQL-client.

5.2 <token> and <separator>

1) *Rationale: <underscore> is already defined in <identifier part>.*

In the Format, replace the production for <identifier body> with:

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

2) *Rationale: <nondelimiter token> must include <binary string literal>.*

In the Format, replace the production for <nondelimiter token> with:

```
<nondelimiter token> ::=
    <regular identifier>
  | <key word>
  | <unsigned numeric literal>
  | <national character string literal>
  | <bit string literal>
  | <hex string literal>
  | <binary string literal>
  | <large object length token>
  | <multiplier>
```

3) *Rationale: Incorrect placement of "see Syntax Rules".*

In the Format, replace the productions for <bracketed comment> and <bracketed comment contents> with:

```
<bracketed comment> ::=
    <bracketed comment introducer>
      <bracketed comment contents>
    <bracketed comment terminator>

<bracketed comment contents> ::= !! See the Syntax Rules
  [ { <comment character> | <separator> }... ]
```

4. Rationale: Correct the BNF of <non-reserved word> and <reserved word>.

In the Format, replace the productions for <non-reserved word> and <reserved word> with:

```
<non-reserved word> ::=
    ADA | ADMIN | ASSIGNMENT | ATTRIBUTE

  | C| CATALOG_NAME | CHAIN | CHARACTER_SET_CATALOG | CHARACTER_SET_NAME
  | CHARACTER_SET_SCHEMA | CHARACTERISTICS | CHECKED | CLASS_ORIGIN
  | COBOL | COLLATION_CATALOG | COLLATION_NAME
  | COLLATION_SCHEMA | COLUMN_NAME | COMMAND_FUNCTION
  | COMMAND_FUNCTION_CODE | COMMITTED | CONDITION_NUMBER
  | CONNECTION_NAME | CONSTRAINT_CATALOG | CONSTRAINT_NAME
  | CONSTRAINT_SCHEMA | CONTAINS | CURSOR_NAME

  | DATETIME_INTERVAL_CODE | DATETIME_INTERVAL_PRECISION | DEFINED
  | DEFINER | DERIVED | DISPATCH

  | FINAL | FORTRAN

  | G | GENERATED | GRANTED

  | HIERARCHY

  | IMPLEMENTATION | INSTANCE | INSTANTIABLE | INVOKER

  | K | KEY_MEMBER | KEY_TYPE

  | LENGTH

  | M | MESSAGE_LENGTH | MESSAGE_OCTET_LENGTH | MESSAGE_TEXT
  | MORE | MUMPS

  | NAME | NULLABLE | NUMBER

  | OPTIONS | ORDERING | OVERRIDING

  | PASCAL | PARAMETER_MODE | PARAMETER_NAME | PARAMETER_ORDINAL_POSITION
  | PARAMETER_SPECIFIC_CATALOG | PARAMETER_SPECIFIC_NAME
  | PARAMETER_SPECIFIC_SCHEMA | PLACING | PLI

  | SCALE | SCHEMA_NAME | SECURITY | SELF | SERIALIZABLE | SERVER_NAME
  | SIMPLE | SOURCE | SPECIFIC_NAME | STATEMENT | STYLE
  | SUBCLASS_ORIGIN

  | TABLE_NAME | TRANSACTION_ACTIVE | TRANSACTIONS_COMMITTED
  | TRANSACTIONS_ROLLED_BACK | TRANSFORM | TRANSFORMS
  | TRIGGER_CATALOG | TRIGGER_NAME | TRIGGER_SCHEMA | TYPE

  | UNCOMMITTED | UNNAMED

<reserved word> ::=
    ABS | ABSOLUTE | ACTION | ADD | AFTER | ALL | ALLOCATE | ALTER | AND
  | ANY | ARE | ARRAY | AS | ASC | ASENSITIVE | ASSERTION | ASYMMETRIC
  | AT | ATOMIC | AUTHORIZATION | AVG
```

```
| BEFORE | BEGIN | BETWEEN | BINARY | BIT | BIT_LENGTH | BLOB | BOOLEAN
| BOTH | BREADTH | BY

| CALL | CALLED | CARDINALITY | CASCADE | CASCADED | CASE | CAST
| CATALOG | CHAR | CHARACTER | CHAR_LENGTH | CHARACTER_LENGTH | CHECK
| CLOB | CLOSE | COALESCE | COLLATE | COLLATION | COLUMN
| COMMIT | CONNECT | CONNECTION | CONSTRAINT | CONSTRAINTS
| CONSTRUCTOR | CONTINUE | CONVERT | CORRESPONDING | COUNT | CREATE
| CROSS | CUBE | CURRENT | CURRENT_DATE | CURRENT_PATH | CURRENT_ROLE
| CURRENT_TIME | CURRENT_TIMESTAMP | CURRENT_USER | CURSOR | CYCLE

| DATA | DATE | DAY | DEALLOCATE | DEC | DECIMAL | DECLARE | DEFAULT
| DEFERRABLE | DEFERRED | DELETE | DEPTH | DEREF | DESC | DESCRIBE
| DESCRIPTOR | DETERMINISTIC | DIAGNOSTICS | DISCONNECT | DISTINCT
| DOMAIN | DOUBLE | DROP | DYNAMIC

| EACH | ELSE | END | END-EXEC | EQUALS | ESCAPE | EVERY | EXCEPT
| EXCEPTION | EXEC | EXECUTE | EXISTS | EXTERNAL | EXTRACT

| FALSE | FETCH | FIRST | FLOAT | FOR | FOREIGN | FOUND | FREE | FROM
| FULL | FUNCTION

| GENERAL | GET | GLOBAL | GO | GOTO | GRANT | GROUP | GROUPING

| HAVING | HOLD | HOUR

| IDENTITY | IMMEDIATE | IN | INDICATOR | INITIALLY | INNER | INOUT
| INPUT | INSENSITIVE | INSERT | INT | INTEGER | INTERSECT | INTERVAL
| INTO | IS | ISOLATION

| JOIN

| KEY

| LANGUAGE | LARGE | LAST | LATERAL | LEADING | LEFT | LEVEL | LIKE
| LOCAL | LOCALTIME | LOCALTIMESTAMP | LOCATOR | LOWER

| MAP | MATCH | MAX | METHOD | MIN | MINUTE | MOD | MODIFIES | MODULE
| MONTH

| NAMES | NATIONAL | NATURAL | NCHAR | NCLOB | NEW | NEXT | NO | NONE
| NOT | NULL | NULLIFY | NUMERIC

| OBJECT | OCTET_LENGTH | OF | OLD | ON | ONLY | OPEN | OPTION | OR
| ORDER | ORDINALITY | OUT | OUTER | OUTPUT | OVERLAPS | OVERLAY

| PAD | PARAMETER | PARTIAL | PATH | PRECISION | POSITION | PREPARE
| PRESERVE | PRIMARY | PRIOR | PRIVILEGES | PROCEDURE | PUBLIC

| READ | READS | REAL | RECURSIVE | REF | REFERENCES | REFERENCING
| RELATIVE | RELEASE | RESTRICT | RESULT | RETURN | RETURNS | REVOKE
| RIGHT | ROLE | ROLLBACK | ROLLUP | ROUTINE | ROW | ROWS

| SAVEPOINT | SCHEMA | SCOPE | SCROLL | SEARCH | SECOND | SECTION
```

```
| SELECT | SENSITIVE | SESSION | SESSION_USER | SET | SETS | SIMILAR
| SIZE | SMALLINT | SOME | SPACE | SPECIFIC | SPECIFICTYPE | SQL
| SQLEXCEPTION | SQLSTATE | SQLWARNING | START | STATE | STATIC
| SUBSTRING | SUM | SYMMETRIC | SYSTEM | SYSTEM_USER

| TABLE | TEMPORARY | THEN | TIME | TIMESTAMP | TIMEZONE_HOUR
| TIMEZONE_MINUTE | TO | TRAILING | TRANSACTION | TRANSLATE
| TRANSLATION | TREAT | TRIGGER | TRIM | TRUE

| UNDER | UNION | UNIQUE | UNKNOWN | UNNEST | UPDATE | UPPER | USAGE
| USER | USING

| VALUE | VALUES | VARCHAR | VARYING | VIEW

| WHEN | WHENEVER | WHERE | WITH | WITHOUT | WORK | WRITE

| YEAR

| ZONE
```

5. *Rationale: Use case-normal forms consistently to define equivalent identifiers.*

Delete NOTE 24.

Delete Syntax Rule 23)

Replace Syntax Rule 24) with:

24) The case- normal form of a <regular 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 corresponding upper- case letter or letters), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_IDENTIFIER.

Replace Syntax Rules 25) and 26) with:

25) Two <regular identifier>s are equivalent if the case-normal forms of their <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 IDC that is sensitive to case, compare equally according to the comparison rules in Subclause 8.2, “<comparison predicate>”.

26) A <regular identifier> and a <delimited identifier> are equivalent if the case-normal form of the <identifier body> of the <regular identifier> 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 collation IDC, compare equally according to the comparison rules in Subclause 8.2, “<comparison predicate>”.

6. *Rationale: <underscore> is already defined in <identifier part>.*

Replace Conformance Rule 1) with:

1) Without Feature F391, "Long identifiers", in a <regular identifier>, the number of <identifier part>s shall be less than 18.

5.3 <literal>

1. Rationale: Clarify the distinction between character sets and character repertoires.

Replace Syntax Rule 7) with:

7) A <nonquote character> is one of:

   a) Any <SQL language character> other than a <quote>;

   b) Any character other than a <quote> in the character set identified by the <module character set specification>; or

   c) Any character other than a <quote> in the character set identified by the <character set specification> or implied by ''N''.

2. Rationale: Clarify the declared type of numeric literals.

Replace Syntax Rules 15) and 16) with:

15) The declared type of an <exact numeric literal> *ENL* is an implementation-defined exact numeric type whose scale is the number of <digit>s to the right of the <period>. There shall be an exact numeric type capable of representing the value of *ENL* exactly.

16) The declared type of an <approximate numeric literal> *ANL* is an implementation-defined approximate numeric type. The value of *ANL* shall not be greater than the maximum value nor less than the minimum value that can be represented by the approximate numeric types. The precise declared type of *ANL* is an implementation-defined approximate numeric type such that the value of *ANL* is not greater than the maximum value that can be represented by the type, nor less than the minimum value that can be represented by the type.

3. Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".

Replace Conformance Rule 7) with:

7) Without Feature F461, "Named character sets", a <character string literal> shall not specify a <character set specification>.

4. Rationale: Editorial.

Replace Conformance Rules 8) and 9) with:

8) Without Feature F411, "Time zone specification", conforming SQL shall not specify a <time zone interval>.

9) Without Feature T041, "Basic LOB data type support", conforming SQL language shall not contain any <binary string literal>.

5.4 Names and identifiers

1. Rationale: Address requirement for multiple diagnostics areas

Replace Syntax Rule 2) with:

2) An <SQL language identifier> is equivalent to an <SQL language identifier> 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 diagnostics areas, and similar uses.

2. Rationale: Use "equivalent" instead of "same" when comparing identifiers.

Replace Syntax Rules 13), 14) and 15) with:

13) Two <schema qualified name>s are equivalent if and only if their <qualified identifier>s are equivalent and their <schema name>s are equivalent, regardless of whether the <schema name>s are implicit or explicit.

13.1) Two <local or schema qualified name>s are equivalent if and only if their <qualified identifier>s are equivalent and either they both specify MODULE or they both specify or imply <schema name>s that are equivalent.

14) Two <character set name>s are equivalent if and only if their <SQL language identifier>s are equivalent and their <schema name>s are equivalent, regardless of whether the <schema name>s are implicit or explicit.

15) Two <schema name>s are equivalent if and only if their <unqualified schema name>s are equivalent and their <catalog name>s are equivalent, regardless of whether the <catalog name>s are implicit or explicit.

3. Rationale: Standardise terminology.

Replace Syntax Rule 19) with:

19) A <collation name> identifies a collation.

4. Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".

Replace Conformance Rule 12) with:

12) Without Feature F461, "Named character sets", conforming SQL language shall not contain any <character set name>.

6.1 <data type>

1. Rationale: Clarify the distinction between character sets and character repertoires.

Delete Syntax Rules 13) and 14).

2. Rationale: Define undefined term used in Part 11.

Insert the following Syntax Rule:

41) If the <data type> *DT1* is contained in a <data type> *DT2* then the *root data type* of *DT1* is the outermost <data type> that contains *DT1*.

3. Rationale: Supply the correct Access Rule for USAGE privileges for types

Replace Access Rule 1) with:

1) If a <user-defined type name>, reference type, <row type> or <collection type> *TY* is specified, and *TY* is usage-dependent on a user-defined type *UDT*, then

Case:

a) If *TY* is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges shall include the USAGE privilege on *UDT*.

b) Otherwise, the current privileges shall include the USAGE privilege on *UDT*.

4. Rationale: Add the notion of a type designator for collection types.

Replace General Rule 8) with:

8) If <data type> is a <collection type>, then a collection type descriptor is created. Let *ETD* be the type designator of the element type of the <collection type>. The collection type descriptor includes the type designator *ETD* ARRAY, an indication of the <collection type constructor> specified by the <collection type> and a descriptor for the element type of the <collection type>.

5. Rationale: Delete a redundant rule.

Delete General Rule 11)

6. Rationale: Add the notion of a type designator for reference types.

Replace General Rule 13) with:

13) If <data type> is a <reference type>, then a reference type descriptor is created. Let *RDTN* be the name of the <referenced type>. The reference type descriptor includes the type designator REF(*RDTN*). If a <scope clause> is specified, then the reference type descriptor includes *STN*, identifying the scope of the reference type.

7. *Rationale: create descriptors for all data types; clarify that certain numeric types may be equivalenced by the SQL-implementation.*

Insert the following General Rules:

14) If <data type> specifies a character string data type, then a character string data type descriptor is created, including the following:

a) The name of the data type (either CHARACTER, CHARACTER VARYING or CHARACTER LARGE OBJECT).

b) The length or maximum length in characters of the character string type.

c) The catalog name, schema name and character set name of the character set of the character string data type.

d) The catalog name, schema name and character set name of the collation of the character string data type.

15) If <data type> is a <binary large object string type>, then a binary string data type descriptor is created, including the following:

a) The name of the data type (BINARY LARGE OBJECT).

b) The maximum length in octets of the binary string data type.

16) If <data type> is a <bit string type>, then a bit string data type descriptor is created, including the following:

a) The name of the data type (BIT or BIT VARYING).

b) The length or maximum length in bits of the bit string type.

17) If <data type> *DT* specifies an exact numeric type, then:

a) There shall be an implementation-defined function *ENNF*() which converts any <exact numeric type> *ENT1* into some possibly different <exact numeric type> *ENT2* (the normal form of *ENT1*), subject to the following constraints on *ENNF*():

i) For every <exact numeric type> *ENT*, *ENNF*(*ENT*) shall not specify DEC or INT.

ii) For every <exact numeric type> *ENT*, the precision, scale and radix of *ENNF*(*ENT*) shall be the precision, scale and radix of *ENT*.

iii) For every <exact numeric type> *ENT*, *ENNF*(*ENT*) shall be the same as *ENNF*(*ENNF*(*ENT*)).

iv) For every <exact numeric type> *ENT*, if *ENNF*(*ENT*) specifies DECIMAL, then *ENNF*(*ENT*) shall specify <precision>, and the precision of *ENNF*(*ENT*) shall be the value of the <precision> specified in *ENNF*(*ENT*).

b) A numeric data type descriptor is created for *DT*, including the following:

i) The name of the type specified in *ENNF*(*DT*) (NUMERIC, DECIMAL, INTEGER, or SMALLINT).

ii) The precision of *DT*.

iii) The scale of *DT*.

iv) An indication of whether the precision and scale are expressed in decimal or binary terms.

18) If <data type> *DT* specifies an approximate numeric type, then:

a) There shall be an implementation-defined function *ANNF*() which converts any <approximate numeric type> *ANT* into some possibly different <approximate numeric type> *ANT2* (the normal form of *ANT1*), subject to the following constraints on *ANNF*():

i) For every <approximate numeric type> *ANT*, the precision of *ANNF*(*ANT*) shall be the precision of *ANT*.

ii) For every <approximate numeric type> *ANT*, *ANNF*(*ANT*) shall be the same as *ANNF*(*ANNF*(*ANT*)).

iii) For every <approximate numeric type> *ANT*, if *ANNF*(*ANT*) specifies FLOAT, then *ANNF*(*ANT*) shall specify <precision>, and the precision of *ANNF*(*ANT*) shall be the value of the <precision> specified in *ANNF*(*ANT*).

b) A numeric data type descriptor is created for *DT* including the following:

i) The name of the type specified in *ANNF*(*DT*) (FLOAT, REAL or DOUBLE PRECISION).

ii) The precision of *DT*.

iii) An indication that the precision is expressed in binary terms.

19) If <data type> specifies <boolean type>, then a boolean data type descriptor is created, including the name of the boolean type (BOOLEAN).

20) If <data type> specifies a <datetime type>, then a datetime data type descriptor is created, including the following:

a) The name of the datetime type (DATE, TIME WITHOUT TIME ZONE, TIME WITH TIME ZONE, TIMESTAMP WITHOUT TIME ZONE, or TIMESTAMP WITH TIME ZONE).

b) The value of the <time fractional seconds precision>, if DATE is not specified.

21) If <data type> specifies an <interval type>, then an interval data type descriptor is created, including the following:

a) The name of the interval data type (INTERVAL).

b) An indication of whether the interval data type is a year-month interval or a day-time interval.

c) The <interval qualifier> simply contained in the <interval type>.

8. Rationale: Use the correct BNF non-terminal (<user-defined type> rather than <user-defined type name>).

Replace Conformance Rule 1) with:

1) Without Feature S023, "Basic structured types", <user-defined type>, if specified, shall not identify a structured type.

9. Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".

Replace Conformance Rule 8) with:

8) Without Feature F461, "Named character sets", a <data type> shall not specify CHARACTER SET.

6.3 <value specification> and <target specification>

1. Rationale: Editorial.

Replace Syntax Rule 7) with:

7) A <target specification> or <simple target specification> that is a <column reference> shall be a new transition variable column reference.
NOTE 65 — "New transition variable column reference" is defined in Subclause 6.5, "<identifier chain>".

8) Let *X* denote either a column *C* or the <key word> VALUE. Given a <boolean value expression> *BVE* and *X*, the notion "*BVE* is a known-not-null condition for *X*" is defined recursively as follows:

a) If *BVE* is a <predicate>, then

Case:

i) If *BVE* is a <predicate> of the form "*RVE* IS NOT NULL", where *RVE* is a <row value expression> that simply contains a <row value constructor element> that is a <column reference> that references *C*, then *BVE* is a *known-not-null condition* for *C*.

ii) If *BVE* is the <predicate> "VALUE IS NOT NULL", then *BVE* is a *known-not-null condition* for VALUE.

iii) Otherwise, *BVE* is not a known-not-null condition for *X*.

b) If *BVE* is a <value expression primary>, then

Case:

i) If *BVE* is of the form "<left paren> <value expression> <right paren>" and the <value expression> is a known-not-null condition for *X*, then *BVE* is a *known-not-null condition* for *X*.

ii) Otherwise, *BVE* is not a known-not-null condition for *X*.

c) If *BVE* is a <boolean test>, then let *BP* be the <boolean primary> immediately contained in *BVE*. If *BP* is a known-not-null condition for *X*, and <truth value> is not specified, then *BVE* is a *known-not-null condition* for *X*. Otherwise, *BVE* is not a known-not-null condition for *X*.

d) If *BVE* is of the form "NOT *BT*", where *BT* is a <boolean test>, then

Case:

i) If *BT* is "*CR* IS NULL", where *CR* is a column reference that references column *C*, then *BVE* is a *known-not-null condition* for *C*.

ii) If *BT* is "VALUE IS NULL", then *BVE* is a *known-not-null condition* for VALUE.

iii) Otherwise, *BVE* is not a known-not-null condition for *X*.

NOTE 66 — For simplicity, the rules do not attempt to analyze conditions such as "NOT NOT A IS NULL", or "NOT (A IS NULL OR NOT (B = 2))"

e) If *BVE* is of the form "*BVE1* AND *BVE2*", then

Case:

i) If either *BVE1* or *BVE2* is a known-not-null condition for *X*, then *BVE* is a *known-not-null condition* for *X*.

ii) Otherwise, *BVE* is not a known-not-null condition for *X*.

f) If *BVE* is of the form "*BVE1* OR *BVE2*", then *BVE* is not a known-not-null condition for *X*.

NOTE 67 — For simplicity, this rule does not detect cases such as "A IS NOT NULL OR A IS NOT NULL", which might be classified as a known-not-null condition.

2. *Rationale: Add missing Conformance Rule.*

Insert the following Conformance Rule:

5) Without Feature S241, "Transform functions", conforming SQL language shall not specify CURRENT_DEFAULT_TRANSFORM_GROUP or CURRENT_TRANSFORM_GROUP_FOR_TYPE.

3. *Rationale: Supply missing Conformance Rule*

Insert the following Conformance Rule:

4.1) Without Feature S241, "Transform functions", conforming SQL language shall not specify CURRENT_DEFAULT_TRANSFORM_GROUP or CURRENT_TRANSFORM_GROUP_FOR_TYPE.

6.5 <identifier chain>

1. *Rationale: Take query names into account and allow for multiple columns of the same name, and provide rules to recognize cursor result column references in the <order by clause> of a cursor.*

Replace Syntax Rule 7) a) with:

7) a) If N = 1 (one), then

Case:

i) If *IC* is contained in an <order by clause> of a <cursor specification>, and the <select list> simply contained in the <cursor specification> directly contains a <derived column> *DC* whose explicit or implicit <column name> is equivalent to *IC*, then PIC_1 is a candidate basis, the scope of PIC_1 is the <cursor specification>, and the referent of PIC_1 is the column referenced by *DC*.

ii) Otherwise, *IC* shall be contained within the scope of one or more exposed <table or query name>s or <correlation name>s whose associated tables include a column whose <identifier> is equivalent to I_1 or within the scope of a <routine name> whose associated <SQL parameter declaration list> includes an SQL parameter whose <identifier> is equivalent to I_1. Let the phrase *possible scope tags* denote those exposed <table or query name>s, <correlation name>s, and <routine name>s.

Case:

1) If the number of possible scope tags in the innermost scope containing a possible scope tag is 1 (one), then let *IPST* be that possible scope tag.

Case:

A) If *IPST* is a <table or query name> or <correlation name>, then let *T* be the table associated with the possible scope tag. For every column *C* of *T* whose <identifier> is equivalent to I_1. PIC_1 is a candidate basis of *IC*, the scope of PIC_1 is the scope of *T*, and the referent of PIC_1 is *C*.

B) If the innermost possible scope tag is a <routine name>, then let *SP* be the SQL parameter whose <identifier> is equivalent to I_1. *PIC1* is the basis of *IC*, the basis length is 1 (one), the basis scope is the scope of *SP*, and the basis referent is *SP*.

2) Otherwise, each possible scope tag shall be a <table or query name> or a <correlation name> of a <table reference> that is directly contained in a <joined table> *JT*. I_1 shall be a common column name in *JT*. Let *C* be the column of *JT* that is identified by I_1. PIC_1 is a candidate basis of *IC*, the scope of PIC_1 is the scope of *T*, and the referent of PIC_1 is *C*

NOTE 72 — "Common column name" is defined in Subclause 7.7, "<joined table>".
NOTE 72.1 — Two or more columns with equivalent column names are distinguished by their ordinal positions within *T*.

2. *Rationale: Allow for multiple columns of the same name.*

Replace Syntax Rule 7) b) with:

7) b) If $N > 1$ (one), then the basis, basis length, basis scope, and basis referent are defined in terms of a candidate basis as follows:

i) If *IC* is contained within the scope of a <routine name> whose associated <SQL parameter declaration list> includes an SQL parameter *SP* whose identifier is equivalent to I_1, then PIC_1 is a candidate basis of *IC*, the scope of PIC_1 is the scope of *SP*, and the referent of PIC_1 is *SP*.

ii) If $N = 2$ and PIC_1 is equivalent to an exposed <correlation name> that is in scope, then let *EN* be the exposed <correlation name> that is equivalent to PIC_1 and has innermost scope. For every column *C*, in the table associated with *EN*, whose <identifier> is equivalent to I_2, PIC_2 is a candidate basis of *IC*, the scope of PIC_2 is the scope of *EN*, and the referent of PIC_2 is *C*.

iii) If $N > 2$ and PIC_1 is equivalent to an exposed <correlation name> that is in scope, then let *EN* be the exposed <correlation name> that is equivalent to PIC_1 and has innermost scope. For every refinable column *C*, in the table associated with *EN*, whose <column name> is equivalent to I_2, PIC_2 is a candidate basis of *IC*, the scope of PIC_2 is the scope of *EN*, and the referent of PIC_2 is *C*.

iv) If $N = 2$, 3 or 4, and if PIC_{N-1} is equivalent to an exposed <table or query name> that is in scope, then let *EN* be the exposed <table or query name> that is equivalent to PIC_{N-1} and has the innermost scope. For every column *C*, in the table associated with *EN*, whose <column name> is equivalent to I_N, PIC_N is a candidate basis of *IC*, the scope of PIC_N is the scope of *EN*, and the referent of PIC_N is *C*.

3. *Rationale: Allow for multiple columns of the same name.*

Replace Syntax Rule 7) c) with:

7) c) There shall be exactly one candidate basis *CB* with innermost scope. The basis of *IC* is *CB*. The basis length is the length of *CB*. The basis scope is the scope of *CB*. The referent of *IC* is the referent of *CB*.

4. *Rationale: Editorial. Add missing text.*

Replace Syntax Rule 11) with:

11) A <basic identifier chain> whose basis referent is a column is a *column reference*. If the basis length is 2, and the basis scope is a <trigger definition> whose <trigger action time> is BEFORE, and I_1 is equivalent to the <new values correlation name> of the <trigger definition>, then the column reference is a *new transition variable column reference*.

6.6 <column reference>

1. *Rationale: clarify how the declared type of a column reference to a column defined using a domain is determined.*

Replace Syntax Rule 5) with:

5) Let *C* be the column that is referenced by *CR*. The declared type of *CR* is

Case:

a) If the column descriptor of *C* includes a data type, then that data type.

b) Otherwise, the data type identified in the domain descriptor included in the column descriptor of *C*.

2. *Rationale: Clarify the semantics of column references.*

Insert the following Syntax Rules:

7) A column reference contained in a <query specification> or a <joined table> is a *queried column reference*.

8) If *QCR* is a queried column reference, then:

a) The *qualifying query* of *QCR* is defined as follows:

i) If *QCR* is contained without an intervening <query specification> in a <joined table> *JT* that is a <query primary>, then *JT* is the qualifying query of *QCR*.

ii) Otherwise, the <query specification> that simply contains the <from clause> that simply contains the <table reference> that defines the qualifying table of *QCR* is the qualifying query of *QCR*.

b) Let *QQ* be the qualifying query of *QCR*.

Case:

i) If *QQ* is a <joined table>, or if *QQ* is not grouped, or if *QCR* is contained in the <where clause> simply contained in *QQ*, then *QCR* is an *ordinary column reference*.

ii) If *QCR* is contained in the <having clause>, <window clause> or <select list> simply contained in *QQ*, and *QCR* is contained in an aggregated argument of a <set function specification> *SFS*, and *QQ* is the aggregation query of *SFS*, then *QCR* is a *within-group varying column reference*.

iii) Otherwise, *QCR* is a *group-invariant column reference*.

9) If *QCR* is a group-invariant column reference, then *QCR* shall be functionally dependent on the grouping columns of the qualifying query of *QCR*.

3. *Rationale: There is no <row value expression> immediately contained in a <set clause>.*

Replace Access Rule 1) with:

1) If *CR* is a <column reference> whose qualifying table is a base table or a viewed table and that is contained in any of:

— A <query expression> simply contained in a <cursor specification>, a <view definition> or an <insert statement>.

— A <sort specification list> contained in a <cursor specification>.

— A <table expression> immediately contained in a <select statement: single row>.

— A <search condition> immediately contained in a <trigger definition>, a <delete statement: searched> or an <update statement: searched>.

— A <select list> immediately contained in a <select statement: single row>.

— A <value expression> simply contained in an <update source>.

then let *C* be the column referenced by *CR*.

Case:

a) If <column reference> is contained in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include SELECT for *C*.

b) Otherwise, the current privileges shall include SELECT on *C*.

NOTE 74 – ''applicable privileges'' and ''current privileges'' are defined in Subclause 10.5, ''<privileges>''.

4. *Rationale: Clarify the semantics of column references.*

Replace the General Rules with:

1) Let *QCR* be a queried column reference. Let *QT* be the qualifying table of *QCR*, and let *C* be the column of *QT* that is referenced as the basis referent of *QCR*. The value of *QCR* is determined as follows:

a) If *QCR* is an ordinary column reference, then *QCR* denotes the value of *C* in a given row of *QT*.

b) If *QCR* is a within-group-varying column reference, then *QCR* denotes the values of *C* in the rows of a given group of the qualifying query of *QCR* used to construct the argument source of a <set function specification>.

c) If *QCR* is a group-invariant column reference, then *QCR* denotes a value that is not distinct from the value of *C* in every row of a given group of the qualifying query of *QCR*. If the most specific type of *QCR* is character string, datetime with time zone, or user-defined type, then the precise value is chosen in an implementation-dependent fashion.

6.8 <field reference>

1. *Rationale: Allow for field names not being unique.*

Replace Syntax Rule 4) with:

4) *FN* shall unambiguously reference a field of *RT*. Let *F* be that field.

6.11 <method invocation>

1. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

In the Format insert the following production:

<constructor method selection> ::= <routine invocation>

2. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Delete Syntax Rule 3).

3. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 5) with:

5) Case:

a) If <method invocation> is immediately contained in <new invocation>, then let *TP* be an SQL-path containing the <schema name> of schema that includes the descriptor of *UDT*.

b) Otherwise, let *TP* be an SQL-path, arbitrarily defined, containing the <schema name> of every schema that includes a descriptor of a supertype or subtype of *UDT*.

4. *Rationale: Provide correct logical structure.*

Replace Syntax Rule 6) with

6) Case:

a) If <generalized invocation> is specified, then let *DT* be the <data type> simply contained in the <generalized invocation>. Let *RI* be the following <method selection>:

MN (*VEP* AS *DT AL*)

b) Otherwise,

Case:

i) If <method invocation> is immediately contained in <new invocation>, then let *RI* be the following <constructor method selection>:

```
MN ( VEP AL )
```

ii) Otherwise, let *RI* be the following <method selection>:

```
MN ( VEP AL )
```

6.14 <dereference operation>

1. Rationale: Editorial corrections

Replace Syntax Rule 2) with:

2) Let *AN* be the <attribute name>. *AN* shall identify an attribute, *AT*, of RT.

Replace Syntax Rule 3) with:

3) The declared type of the <dereference operation> is the declared type of *AT*.

Replace Syntax Rule 4) with:

4) Let *S* be the name of the referenceable table in the scope of the reference type of *RVE*.

a) Let *OID* be the name of the self-referencing column of *S*.

b) <dereference operation> is equivalent to a <scalar subquery> of the form:

```
( SELECT AN
    FROM S
   WHERE S.OID = RVE)
```

6.16 <set function specification>

1. Rationale: Permit outer references in a <select list>.

Insert the following Syntax Rule:

3.1) The <value expression> simply contained in <set function specification> *SFS* is an *aggregated argument* of *SFS*.

3.2) A column reference *CR* contained in an aggregated argument of a <set function specification> *SFS* is called an *aggregated column reference* of *SFS*.

2. Rationale: Permit outer references in a <select list>.

Insert the following Syntax Rules:

5.1) The *aggregation query* of a <set function specification> *SFS* is determined as follows:

a) If *SFS* has no aggregated column reference, then the aggregation query of *SFS* is the innermost <query specification> that contains *SFS*.

b) Otherwise, the innermost qualifying query of the aggregated column references of *SFS* is the aggregation query of *SFS*.

5.2) *SFS* shall be contained in the <having clause>, <window clause> or <select list> of its aggregation query.

5.3) Let *CR* be an aggregated column reference of *SFS* such that the qualifying query *QQ* of *CR* is not the aggregation query of *SFS*. If *QQ* is grouped and *SFS* is contained in the <having clause>, <window clause> or <select list> of *QQ*, then *CR* shall be functionally dependent on the grouping columns of *QQ*.

3. *Rationale: Disallow DISTINCT on columns with types based on LOB and array types. Provide consistent Syntax Rules for comparison operations.*

Insert the following Syntax Rule:

6.1) If the <set function specification> specifies a <general set function> whose <set quantifier> is DISTINCT, then *DT* shall not be LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

Delete Syntax Rule 7).

4. *Rationale: Provide consistent Syntax Rules for comparison operations.*

Replace Syntax Rule 8) with:

8) If the <set function specification> specifies a <set function type> that is MAX or MIN, then DT shall not be LOB-ordered, array-ordered, reference-ordered, UDT-EC-ordered, UDT-NC-ordered, or a row type.

Delete Syntax Rule 10).

5. *Rationale: Clarify the declared type of numeric set functions.*

Replace Syntax Rule 12) with:

12) If COUNT is specified, then the declared type of the result is an implementation-defined exact numeric type with scale of 0 (zero).

Replace Syntax Rule 14) with:

14) If SUM or AVG is specified, then:

a) *DT* shall be a numeric type or an interval type.

b) If SUM is specified and *DT* is exact numeric with scale *S*, then the declared type of the result is an implementation-defined exact numeric type with scale *S*.

c) If AVG is specified and *DT* is exact numeric, then the declared type of the result is an implementation-defined exact numeric type with precision not less than the precision of *DT* and scale not less than the scale of *DT*.

d) If *DT* is approximate numeric, then the declared type of the result is an implementation-defined approximate numeric type with precision not less than the precision of *DT*.

e) If *DT* is interval, then the declared type of the result is interval with the same precision as *DT*.

6. *Rationale: Changes to <group by clause> have changed the treatment of <grouping operation>. Editorial - Erroneous text not deleted.*

Replace General Rules 3) h) and 3) i) with:

NOTE 78.1 — the value of <grouping operation> is computed by means of syntactic transformations in 7.9 "<group by clause>".

7. *Rationale: Correcting the Feature Name of Feature T431 in the Conformance Rules of <set function specification>.*

Replace Conformance Rule 7) with:

7) Without Feature T431, "Extended grouping capabilities", conforming SQL language shall not contain a <set function specification> that is a <grouping operation>.

8. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Replace Conformance Rule 8) with:

8) Without Feature S024, "Enhanced structured types", in a <general set function>, if MAX or MIN is specified, then the <value expression> shall not be of an ST-ordered declared type.

Insert the following Conformance Rule:

10) Without Feature S024, "Enhanced structured types", if a <set qualifier> of DISTINCT is specified, then the <value expression> shall not be of ST-ordered declared type.

6.17 <numeric value function>

1. *Rationale: Clarify the declared type of <numeric value function>.*

Replace Syntax Rule 1) with:

1) If <position expression> is specified, then the declared type of the result is an implementation-defined exact numeric type with scale 0 (zero).

Replace Syntax Rule 3) with:

3) If <extract expression> is specified, then

Case:

a) If <primary datetime field> does not specify SECOND, then the declared type of the result is an implementation-defined exact numeric type with scale 0 (zero).

b) Otherwise, the declared type of the result is an implementation-defined exact numeric type with scale not less than the specified or implied <time fractional seconds precision> or <interval fractional seconds precision>, as appropriate, of the SECOND <primary datetime field> of the <extract source>.

Delete Syntax Rules 4) and 5).

Replace Syntax Rules 6), 7), 8), 9), 10), 11) and 12) with:

6) If a <length expression> is specified, then the declared type of the result is an implementation-defined exact numeric type with scale 0 (zero).

7) If <cardinality expression> is specified, then the declared type of the result is an implementation-defined exact numeric type with scale 0 (zero).

8) If <absolute value expression> is specified, then the declared type of the result is the declared type of the immediately contained <numeric value expression>.

9) If <modulus expression> is specified, then the declared type of each <numeric value expression> shall be exact numeric with scale 0 (zero). The declared type of the result is the declared type of the immediately contained <numeric value expression divisor>.

Delete Syntax Rules 10), 11) and 12).

6.18 <string value function>

1. Rationale: Provide more meaningful keywords for the <regular expression substring function>.

In the Format, replace the production for <regular expression substring function> with:

```
<regular expression substring function> ::=
     SUBSTRING <left paren> <character value expression>
     SIMILAR <character value expression>
     ESCAPE <escape character> <right paren>
```

2. Rationale: Editorial. Standardise terminology.

Replace Syntax Rule 2) with:

2) The declared type, coercibility, and collation of <character value function> are the declared type, coercibility, and collation, respectively, of the immediately contained <character substring function>, <regular expression substring function>, <fold>, <form-of-use conversion>, <character translation>, <trim function>, <specific type method>, or <character overlay function>.

3. Rationale: A <character value function> that operates on a <character value expression> of character large object should return character large object. Standardise terminology.

Replace Syntax Rule 4) with:

4) If <character substring function> is specified, then:

a) Case:

i) If the declared type of <character value expression> is fixed-length character string or variable-length character string, then the declared type of the <character substring function> is variable-length character string with maximum length equal to the fixed length or maximum length of the <character value expression>.

ii) Otherwise, the declared type of the <character substring function> is large object character string with a maximum length equal to the maximum length of the <character value expression>.

a.1) The character repertoire and form-of-use of the <character substring function> are the same as the character repertoire and form-of-use of the <character value expression>.

b) The collation and the coercibility characteristic are determined as specified for monadic operators in Subclause 4.2.3, "Rules determining collating sequence usage", where the first operand of <character substring function> plays the role of the monadic operand.

4. *Rationale: A <character value function> that operates on a <character value expression> of character large object should return character large object. Standardise terminology.*

Replace Syntax Rule 5) with:

5) If <regular expression substring function> is specified, then:

a) The declared types of the <escape character> and the <character value expression>s of the <regular expression substring function> shall be character string with the same character repertoire.

b) Case:

i) If the declared type of <character value expression> is fixed-length character string or variable-length character string, then the declared type of the <regular expression substring function> is variable-length character string with maximum length equal to the maximum length of the first <character value expression>.

ii) Otherwise, the declared type of the <regular expression substring function> is large object character string with a maximum length equal to the maximum length of the first <character value expression>.

b.1) The character repertoire and form-of-use of <regular expression substring function> are the same as the character repertoire and form-of-use of the first <character value expression>.

c) The value of the <escape character> shall have length 1 (one).

d) The collation and the coercibility characteristic are determined as specified for monadic operators in Subclause 4.2.3, "Rules determining collating sequence usage", where the first operand of <regular expression substring function> plays the role of the monadic operand.

5. *Rationale: A <character value function> that operates on a <character value expression> of character large object should return character large object. Clarify the distinction between character sets and character repertoires.*

Replace Syntax Rule 7) with:

7) If <form-of-use conversion> is specified, then:

a) <form-of-use conversion> shall be simply contained in a <value expression> that is immediately contained in a <derived column> that is immediately contained in a <select sublist> or shall immediately contain either a <simply value specification> that is a <host parameter name> or a <value specification> that is a <host parameter specification>.

b) A <form-of-use conversion name> shall identify a form-of-use conversion.

c) Case:

i) If the declared type of <character value expression> is fixed-length character string or variable-length character string, then the declared type of the result is variable-length character string with implementation-defined maximum length.

ii) Otherwise, the declared type of the result is large object character string with implementation-defined maximum length.

c.1) The character set of the result is the character set *CS* whose character repertoire is the same as the character repertoire of the <character value expression> and character encoding form is that determined by the form-of-use conversion identified by the <form-of-use conversion name>. The result has the *Implicit* coercibility characteristic and its collation is the character set collation of *CS*.

6. *Rationale: A <character value function> that operates on a <character value expression> of character large object should return character large object. Clarify the distinction between character sets and character repertoires.*

Replace Syntax Rule 8) with:

8) If <character translation> is specified, then:

a) A <translation name> shall identify a character translation.

b) Case:

i) If the declared type of <character value expression> is fixed-length character string or variable-length character string, then the declared type of the <character translation> is variable-length character string with implementation-defined maximum length.

ii) Otherwise, the declared type of the <character translation> is large object character string with implementation-defined maximum length.

b.1) The declared type of the <character translation> has a character set *CS* that is the target character set of the translation. The result has the *Implicit* coercibility characteristic and its collation is the character set collation of *CS*.

7. *Rationale: A <character value function> that operates on a <character value expression> of character large object should return character large object. Standardise terminology.*

Replace Syntax Rule 9) with:

9) If <trim function> is specified, then;

a) If FROM is specified, then:

i) Either <trim specification> or <trim character> or both shall be specified.

ii) If <trim specification> is not specified, then BOTH is implicit.

iii) If <trim character> is not specified, then ' ' is implicit.

b) Otherwise, let *SRC* be <trim source>.

```
TRIM ( SRC )
```

is equivalent to

```
TRIM ( BOTH ' ' FROM SRC )
```

c) Case:

i) If the declared type of <character value expression> is fixed-length character string or variable-length character string, then the declared type of the <trim function> is variable-length character string with maximum length equal to the fixed length or maximum length of the <trim source>.

ii) Otherwise, the declared type of the <trim function> is large object character string with maximum length equal to the maximum length of the <trim source>.

d) If a <trim character> is specified, then <trim character> and <trim source> shall be comparable.

e) The character repertoire and form-of-use of the <trim function> are the same as those of the <trim source>.

f) The collation and the coercibility characteristic are determined as specified for monadic operators in Subclause 4.2.3, "Rules determining collating sequence usage", where the <trim source> plays the role of the monadic operand.

8. *Rationale: Clarify the algorithm for <regular expression substring function>.*

Replace General Rules 4)c), 4)d) and 4)e) with:

4) c) If the length in characters of *E* is not equal to 1 (one), then an exception condition is raised: *data exception — invalid escape character.*

d) If *R* does not contain exactly two occurrences of the two-character sequence consisting of *E* followed by the <double quote> character, then an exception condition is raised: *data exception — invalid use of escape character.*

e) Let *R1*, *R2*, and *R3* be the substrings of *R*, such that

```
'R' = 'R1' || 'E' || '"' || 'R2' || 'E' || '"' || 'R3'
```

is *True*.

f) If any one of *R1*, *R2* or *R3* is not a zero-length string and does not have the format of a <regular expression>, then an exception condition is raised: *data exception — invalid regular expression*.

g) If the predicate

```
'C' SIMILAR TO '(R1)' || '(R2)' || '(R3)' ESCAPE 'E'
```

is not *True*, then the result of the <regular expression substring function> is the null value.

h) Otherwise, the result of the <regular expression substring function> is computed as follows:

i) Let *S1* be the shortest initial substring of *C* such that there is a sub-string *S23* of *C* such that the following <search condition> is *True*:

```
'C' = 'S1' || 'S23' AND
'S1' SIMILAR TO 'R1' ESCAPE 'E' AND
'S23' SIMILAR TO '(R2)(R3)' ESCAPE 'E'
```

ii) Let *S3* be the shortest final substring of *S23* such that there is a sub-string *S2* of *S23* such that the following <search condition> is *True*:

```
'S23' = 'S2' || 'S3' AND
'S2' SIMILAR TO 'R2' ESCAPE 'E' AND
'S3' SIMILAR TO 'R3' ESCAPE 'E'
```

iii) The result of the <regular expression substring function> is *S2*.

9. Rationale: Editorial.

Replace General Rule 5) c) ii) with:

5) c) ii) If LOWER is specified, then the result of the <fold> is a copy of *S* in which every upper case character that has a corresponding lower case character or characters in the character set of *S* and every title case character that has a corresponding lower case character or characters in the character set of *S* is replaced by that lower case character or characters.

10. Rationale: Remove ambiguity caused by the fact that the result of case folding may be of a different length in characters than the source string.

Replace General Rule 5) with:

5) If <fold> is specified, then:

a) Let *S* be the value of the <character value expression>.

b) If *S* is the null value, then the result of the <fold> is the null value.

c) Let *FRML* be the length or the maximum length in characters of the declared type of <fold>.

d) Case:

i) If UPPER is specified, then let *FR* be a copy of *S* in which every lower case character that has a corresponding upper case character or characters in the character set of *S* and every title case character that has a corresponding upper case character or characters in the character set of *S* is replaced by that upper case character or characters.

ii) If LOWER is specified, then let *FR* be a copy of *S* in which every upper case character that has a corresponding lower case character or characters in the character set of *S* and every title case character that has a corresponding lower case character or characters in the character set of *S* is replaced by that lower case character or characters.

e) Let *FRL* be the length in characters of *FR*.

f) Case:

i) If *FRL* is less than or equal to *FRML*, then the result of the <fold> is *FR*.

ii) If *FRL* is greater than *FRML*, then the result of the <fold> is the first *FRML* characters of *FR*. If the declared type of *FR* is fixed length, then the result is padded on the right with (*FRML* - *FRL*) <space>s. If any of the right-most *FRL* - *FRML* characters of *FR* are non-<space> characters, then a completion condition is raised: *warning — string data, right truncation*.

11. Rationale: Double double quotes when constructing delimited identifier string .

Replace General Rule 9) with:

9) If <specific type method> is specified, then:

a) Let *V* be the value of the <user-defined type value expression>.

b) Case:

i) If *V* is the null value, then *RV* is the null value.

ii) Otherwise:

1) Let *UDT* be the most specific type of *V*.

2) Let *UDTN* be the <user-defined type name> of *UDT*.

3) Let *CN* be the <catalog name> contained in *UDTN*, let *SN* be the <unqualified schema name> contained in *UDTN*, and let *UN* be the <qualified identifier> contained in *UDTN*. Let *CND*, *SND* and *UND* be *CN*, *SN* and *UN*, respectively, with every occurrence of <double quote> replaced by <doublequote symbol>. Let *RV* be:

"*CND*"."*SND*"."*UND*"

c) The result of <specific type method> is *RV*.

12. Rationale: Editorial.

Replace Conformance Rule 3) with:

3) Without Feature T042, "Extended LOB data type support", conforming SQL language shall not contain any <blob value function>.

6.19 <datetime value function>

1. Rationale: Correct the use of general containment

Replace General Rule 3) with:

3) Let *S* be an <SQL procedure statement> that is not generally contained in a <triggered action>. All <datetime value function>s that are contained in <value expression>s that are generally contained, without an intervening <routine invocation> whose subject routines do not include an SQL function, either in *S* without an intervening <SQL procedure statement> or in an <SQL procedure statement> contained in the <triggered action> of a trigger activated as a consequence of executing *S*, are effectively evaluated simultaneously. The time of evaluation of a <datetime value function> during the execution of S and its activated triggers is implementation-dependent.

6.20 <interval value function>

1. Rationale: Editorial - typographical error.

Replace Syntax Rule 1) with:

1) If <interval absolute value function> is specified, then the declared type of the result is the declared type of the <interval value expression>.

Replace General Rule 1) with:

1) If <interval absolute value function> is specified, then let *N* be the value of the <interval value expression>.

Case:

a) If *N* is the null value, then the result is the null value.

b) If $N \geq 0$ (zero), then the result is *N*.

c) Otherwise, the result is -1***N*.

2. Rationale: Editorial.

Replace Conformance Rule 1) with:

1) Without Feature F052, "Intervals and datetime arithmetic", conforming SQL shall contain no <interval value function>.

6.22 <cast specification>

1. Rationale: Editorial.

Replace Syntax Rule 4) with:

4) Let *C* be some column and let *CO* be the <cast operand> of a <cast specification> *CS*. *C* is a leaf column of *CS* if *CO* consists of a single column reference that identifies *C* or of a single <cast specification> *CS1* of which *C* is a leaf column.

2. Rationale: Editorial.

Replace the introductory paragraph of Syntax Rule 6) with:

6) If the <cast operand> is a <value expression>, then the valid combinations of *TD* and *SD* in a <cast specification> are given by the following table. "Y" indicates that the combination is syntactically valid without restriction; "M" indicates that the combination is valid subject to other Syntax Rules in this Subclause being satisfied; and "N" indicates that the combination is not valid:

3. Rationale: Only check for user-defined casts if <cast operand> is a <value expression>.

Replace Syntax Rule 11) with:

11) If the <cast operand> is a <value expression> and if either *SD* or *TD* is a user-defined type, then there shall be a data type *P* such that:

a) The type designator of *P* is in the type precedence list of *SD*.

b) There is a user-defined cast CF_p whose user-defined cast descriptor includes *P* as the source data type and *TD* as the target data type.

c) The type designator of no other data type *Q* that is included as the source data type in the user-defined cast descriptor of some user-defined cast CF_q that has *TD* as the target data type precedes the type designator of *P* in the type precedence list of *SD*.

NOTE 84 – *Source type* is defined in Subclause 4.8, "User-defined types".

Replace Syntax Rule 12) with:

12) If the <cast operand> is a <value expression> and if either *SD* or *TD* is a reference type, then:

a) Let *RTSD* and *RTTD* be the referenced types of *SD* and *TD*, respectively.

b) If <data type> is specified and contains a <scope clause>, then let *STD* be that scope. Otherwise, let *STD*, possibly empty, be the scope included in the reference type descriptor of *SD*.

c) Either *RSTD* and *RTTD* shall be compatible, or there shall be a data type *P* in the type precedence list of *SD* such that all of the following are satisfied:

i) There is a user-defined cast CF_p whose user-defined cast descriptor includes *P* as the source data type and *TD* as the target data type.

ii) No other data type *Q* that is included as the source data type in the user-defined cast descriptor of some user-defined cast CF_q that has *TD* as the target data type precedes *P* in the type precedence list of *SD*.

4. *Rationale: Editorial.*

Replace Syntax Rule 13) c) with:

13) c) CAST (*VALUE* AS *ETD*)
where *VALUE* is a <value expression> of declared type *ESD*, shall be a valid <cast specification>.

5. *Rationale: Only check for user-defined casts if <cast operand> is a <value expression>.*

Replace Access Rule 2) with:

2) If the <cast operand> is a <value expression> and if either *SD* or *TD* is a user-defined type, then

Case:

a) If <cast specification> is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include EXECUTE on CF_p.

b) Otherwise, the current privileges shall include EXECUTE on CF_p.

NOTE 86 – "applicable privileges" and "current privileges" are defined in Subclause 10.5, "<privileges>".

6. *Rationale: General Rule 3) and all of the following General Rules are only applicable if the <cast operand> is a <value expression> whose value is not null.*

Replace General Rule 2) with:

2) Case:

a) If the <cast operand> specifies NULL, then *TV* is the null value and no further General Rules of this Subclause are applied.

b) If the <cast operand> specifies an <empty specification>, then *TV* is an empty collection of declared type *TD* and no further General Rules of this Subclause are applied.

c) If *SV* is the null value, then *TV* is the null value and no further General Rules of this Subclause are applied.

d) Otherwise, let *TV* be the result of the <cast specification> as specified in the remaining General Rules of this Subclause.

7. *Rationale: Avoid an infinite regress.*

Replace General Rule 3)c) with:

3) c) Case:

i) If *TD* is a user-defined type, then *TV* is *TR*.

ii) Otherwise, *TV* is the result of

CAST (*TR* AS *TD*)

8. Rationale: Standardise terminology.

Replace General Rule 8) a) i) with:

8) a) i) If *Y* contains any <SQL language character> that is not in the character repertoire of *TD*, then an exception condition is raised: *data exception — invalid character value for cast.*

Replace General Rule 8) b) iii) 1) with:

8) b) iii) 1) If *Y* contains any <SQL language character> that is not in the character repertoire of *TD*, then an exception condition is raised: *data exception — invalid character value for cast.*

9. Rationale: Improve wording.

Replace General Rule 8) d) with:

8) 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 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).

If *PAD* is not 0 (zero), then append (*B* - *PAD*) 0-valued bits to the least significant end of *SV*; a completion condition is raised: *warning — implicit zero-bit padding.*

Let *SVC* be the possibly padded value of *SV* regarded as a character string without regard to valid character encodings and let *LTDS* be a character string of *LTD* characters of value *NC* characters in the form-of-use of *TD*.

TV is the result of

```
SUBSTRING ( SVC || LTDS FROM 1 FOR LTD )
```

Case:

i) If the length of *TV* is less than the length of *SVC*, then a completion condition is raised: *warning — string data, right truncation.*

ii) If the length of *TV* is greater than the length of *SVC*, then a completion condition is raised: *warning — implicit zero-bit padding.*

10. Rationale: Standardise terminology.

Replace General Rule 8) e) i) with:

8) e) i) If *Y* contains any <SQL language character> that is not in the character repertoire of *TD*, then an exception condition is raised: *data exception — invalid character value for cast.*

Replace General Rule 9) a) i) with:

9) a) i) If *Y* contains any <SQL language character> that is not in the character repertoire of *TD*, then an exception condition is raised: *data exception — invalid character value for cast.*

Replace General Rule 9) b) iii) 1) with:

9) b) iii) 1) If *Y* contains any <SQL language character> that is not in the character repertoire of *TD*, then an exception condition is raised: *data exception — invalid character value for cast.*

11. Rationale: Improve wording.

Replace General Rule 9) d) with:

9) 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 of *TD*.

Let *PAD* be the value of the remainder of the division *LSV* / *B*.

If *PAD* is not 0 (zero), then append (*B* - *PAD*) 0-valued bits to the least significant end of *SV*; a completion condition is raised: *warning — implicit zero-bit padding.*

Let *SVC* be the possible padded value of *SV* regarded as a character string without regard to valid character encodings.

Case:

i) If CHARACTER_LENGTH(*SVC*) is not greater than *MLTD*, then *TV* is *SVC*.

ii) Otherwise, *TV* is the result of

```
SUBSTRING ( SVC FROM 1 FOR MLTD )
```

If the length of *TV* is less than the length of *SVC*, then a completion condition is raised: *warning — string data, right truncation.*

12. Rationale: Standardise terminology.

Replace General Rule 9) e) i) with:

9) e) i) If *Y* contains any <SQL language character> that is not in the character repertoire of *TD*, then an exception condition is raised: *data exception — invalid character value for cast.*

13. Rationale: Improve wording.

Replace General Rule 11) with:

11) If *TD* is fixed-length bit string, then let *LTD* be the length in bits of *TD*. Let *BLSV* be the result of BIT_LENGTH(*SV*).

Case:

a) If *BLSV* is equal to *LTD*, then *TV* is *SV* regarded as a bit string with a length in bits of *BLSV*.

b) If *BLSV* is larger than *LTD*, then *TV* is the first *LTD* bits of *SV* regarded as a bit string with a length in bits of *LTD*, and a completion condition is raised: *warning — string data, right truncation*.

c) If *BLSV* is smaller than *LTD*, then *TV* is *SV* regarded as a bit string extended on the right with *LTD* - *BLSV* bits whose values are all 0 (zero) and a completion condition is raised: *warning — implicit zero-bit padding*.

Replace General Rule 12) with:

12) If *TD* is variable-length bit string, then let *MLTD* be the maximum length in bits of *TD*. Let *BLSV* be the result of BIT_LENGTH(*SV*).

Case:

a) If *BLSV* is less than or equal to *MLTD*, then *TV* is *SV* regarded as a bit string with a length in bits of *BLSV*.

b) If *BLSV* is larger than *MLTD*, then *TV* is the first *MLTD* bits of *SV* regarded as a bit string with a length in bits of *MLTD* and a completion condition is raised: *warning — string data, right truncation.*

14. Rationale: Editorial.

Replace General Rule 16) d) with:

16) d) If *SD* is TIMESTAMP WITH TIME ZONE, then the UTC component of *TV* is the hour, minute, and second <primary datetime field>s of *SV*, with implementation-defined rounding or truncation if necessary, and the time zone component of *TV* is the time zone displacement of *SV*.

15. Rationale: Editorial.

Replace General Rule 21) with:

21) If the <cast specification> contains a <domain name> and that <domain name> refers to a domain that contains a <domain constraint> and if *TV* does not satisfy the <check constraint definition> of the <domain constraint>, then an exception condition is raised: *integrity constraint violation.*

6.23 <value expression>

1. Rationale: Correct the declared data type and value of <value expression primary>.

Replace Syntax Rule 2) with:

2) The declared type of a <value expression primary> is the declared type of the simply contained <unsigned value specification>, <column reference>, <set function specification>, <scalar subquery>, <case expression>, <value expression>, <cast specification>, <subtype treatment>, <attribute or method reference>, <reference resolution>, <collection value constructor>, <field reference>, <element reference>, <method invocation>, or <static method invocation>, or the effective returns type of the immediately contained <routine invocation>, respectively.

2. Rationale: Define possibly non-deterministic <value expression>

Insert the following Syntax Rule:

9) A <value expression> or <nonparenthesized value expression primary> is *possibly non-deterministic* if it generally contains any of the following:

a) A <datetime value function>.

b) A <cast specification> whose result type is datetime with time zone and whose <cast operand> has declared type that is not datetime with time zone.

c) A <datetime factor> that simply contains a <datetime primary> whose declared type is datetime without time zone and that simply contains an explicit <time zone>.

c.1) An <interval value expression> that computes the difference of a <datetime value expression> and a <datetime term>, such that the declared type of one operand is datetime with time zone and the other operand is datetime without time zone.

c.2) A <comparison predicate> , <overlaps predicate>, or <distinct predicate> simply containing <row value predicand>s *RVP1* and *RVP2* such that the declared types of *RVP1* and *RVP2* have corresponding constituents such that one constituent is datetime with time zone and the other is datetime without time zone.
NOTE 86.1 — This includes <between predicate> because of a syntactic transformation to <comparison predicate>.

c.3) A <quantified comparison predicate> or a <match predicate> simply containing a <row value predicand> *RVP* and a <table subquery> *TS* such that the declared types of *RVP* and *TS* have corresponding constituents such that one constituent is datetime with time zone and the other is datetime without time zone.
NOTE 86.2 — This includes <in predicate> because of a syntactic transformation to <quantified comparison predicate>.

d) A <value specification> that is CURRENT_USER, CURRENT_ROLE, SESSION_USER, SYSTEM_USER, or CURRENT_PATH.

e) A <routine invocation> whose subject routine is an SQL-invoked routine that is possibly non-deterministic.

f) A <set function specification> that specifies MIN or MAX and that simply contains a <value expression> whose declared type is based on a character string type, user-defined type, or datetime with time zone.

g) A <query specification> or <query expression> that is possibly non-deterministic.

h) A <cast specification> whose result type is an array type *CT1* such that the declared element type of *CT1* is datetime with time zone and whose <cast operand> has declared type that is an array type *CT2* such that the declared element type of *CT2* is not datetime with time zone.

3. *Rationale: Correct the declared data type and value of <value expression primary>.*

Replace General Rule 5) with:

5) The value of a <value expression primary> is the value of the simply contained <unsigned value specification>, <column reference>, <set function specification>, <scalar subquery>, <case expression>, <value expression>, <cast specification>, <subtype treatment>, <collection value constructor>, <field reference>, <element reference>, <method invocation>, <static method invocation>, <routine invocation>, or <attribute or method reference>.

6.24 <new specification>

1. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

In the Format, replace the production for <new invocation> with:

```
<new invocation> ::=
      <method invocation>
    | <routine invocation>
```

2. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 3) with:

3) Case:

a) If the <new specification> is of the form

```
NEW RN( )
```

then:

Case:

i) If *S* does not include the descriptor of a constructor method whose method name is equivalent to *MN* and whose unaugmented parameter list is empty, then the <new specification> is equivalent to the <new invocation>

```
RN( )
```

ii) Otherwise, the <new specification> is equivalent to the <new invocation>

```
RN( ).MN( )
```

b) Otherwise, the <new specification>

```
NEW RN( a1, a2, ..., an )
```

is equivalent to the <new invocation>

```
RN().MN( a1, a2, ..., an )
```

3. *Rationale: Editorial.*

Replace Conformance Rule 1) with:

1) Without Feature S023, "Basic structured types", conforming SQL language shall not contain any <new specification>.

6.25 <subtype treatment>

1. *Rationale: Resolve incorrect duplication of BNF non-terminal symbol.*

Replace the Format with:

```
<subtype treatment> ::=
     TREAT <left paren> <subtype operand>
       AS <target subtype> <right paren>

<subtype operand> ::= <value expression>

<target subtype> ::=
     <user-defined type>
```

2. *Rationale: Correctly specify the conditions under which an exception is raised.*

Replace General Rules 2) and 3) with:

2) Case:

   a) If *V* is the null value, then the value of the <subtype treatment> is the null value.

   b) Otherwise,

      i) If the most specific type of *V* is not a subtype of *DT*, then an exception condition is raised: *invalid target type specification.*
      NOTE 107 – ''most specific type'' is defined in Subclause 4.8.3, ''Subtypes and supertypes''.

      ii) The value of the <subtype treatment> is *V*.

3. *Rationale: Editorial.*

Replace Conformance Rule 1) with:

1) Without Feature S161, "Subtype treatment", conforming SQL Language shall contain no <subtype treatment>.

6.26 <numeric value expression>

1. Rationale: Clarify the declared type of <numeric value expression>

Replace Syntax Rule 1) with:

1) If the declared type of both operands of a dyadic arithmetic operator is exact numeric, then the declared type of the result is an implementation-defined exact numeric type, with precision and scale determined as follows:

   a) Let *S1* and *S2* be the scale of the first and second operands respectively.

   b) The precision of the result of addition and subtraction is implementation-defined, and the scale is the maximum of *S1* and *S2*.

   c) The precision of the result of multiplication is implementation-defined, and the scale is *S1*+ *S2*.

   d) The precision and scale of the result of division is implementation-defined.

Replace Syntax Rule 2) with:

2) If the declared type of either operand of a dyadic arithmetic operator is approximate numeric, then the declared type of the result is an implementation-defined approximate numeric type.

2. Rationale: Editorial.

Replace General Rule 5) with:

5) If the most specific type of the result of an arithmetic operation is exact numeric, then

   Case:

   a) If the operator is not division and the mathematical result of the operation is not exactly representable with the precision and scale of the result data type, then an exception condition is raised: *data exception — numeric value out of range.*

   b) If the operator is division and the approximate mathematical result of the operation represented with the precision and scale of the result data type loses one or more leading significant digits after rounding or truncating if necessary, then an exception condition is raised: *data exception — numeric value out of range*. The choice of whether to round or truncate is implementation-defined.

6.27 <string value expression>

1. Rationale: Clarify the distinction between character sets and character repertoires.

Replace Syntax Rule 2) with:

2) Character strings of different character repertoires shall not be mixed in a <character value expression>.

2.1) The character set of a <character value expression> is that character set of its character string operands that has the character encoding form with the highest precedence.

2. *Rationale: Include character large object in the description of the effects of <concatenation>.*

Replace Syntax Rule 3) with:

3) Case:

a) If <concatenation> is specified, then:
Let *D1* be the declared type of the <character value expression> and let *D2* be the declared type of the <character factor>. Let *M* be the length in characters of *D1* plus the length in characters of *D2*. Let *VL* be the implementation-defined maximum length of a variable-length character string, let *LOL* be the implementation-defined maximum length of a large object character string, and let *FL* be the implementation-defined maximum length of a fixed-length character string.

Case:

i) If the declared type of the <character value expression> or <character value expression> or <character factor> is large object character string, then the declared type of the <concatenation> is large object character string with maximum length equal to the lesser of *M* and *LOL*.

ii) If the declared type of the <character value expression> or <character factor> is variable-length character string, then the declared type of the <concatenation> is variable-length character string with maximum length equal to the lesser of *M* and *VL*.

iii) If the declared type of the <character value expression> and <character factor> is fixed-length character string, then *M* shall not be greater than *FL* and the declared type of the <concatenation> is fixed-length character string with length *M*.

b) Otherwise, the declared type of the <character value expression> is the declared type of the <character factor>.

3. *Rationale: Include character large object in the description of the effects of <concatenation>.*

Replace General Rule 2) with:

2) If <concatenation> is specified, then let *S1* and *S2* be the result of the <character value expression> and <character factor>, respectively.

Case:

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

b) Otherwise, let *S* be the string consisting of *S1* followed by *S2* and let *M* be the length of *S*.

Case:

i) 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 a large object character string.

Case:

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

2) If *M* is greater than *LOL* and the right-most *M* 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) Otherwise, an exception condition is raised: *data exception — string data, right truncation.*

ii) 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 a variable-length character string.

Case:

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

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

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

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

6.28 <datetime value expression>

1. Rationale: Reword to avoid problem when the result does not have multiple components.

Replace General Rule 6) with:

6) If a <datetime value expression> immediately contains the operator <plus sign> or <minus sign>, then the time zone component, if any, of the result is the same as the time zone component of the immediately contained <datetime term> or <datetime value expression>. The result (if the result type is without time zone) or the UTC component of the result (if the result type is with time zone) is effectively evaluated as follows:

a) Case:

i) If <datetime value expression> immediately contains the operator <plus sign> and the <interval value expression> or <interval term> is not negative, or if <datetime value expression> immediately contains the operator <minus sign> and the <interval term> is negative, then successive <primary datetime field>s of the <interval value expression> or <interval term> are added to the corresponding fields of the <datetime value expression> or <datetime term>.

ii) Otherwise, successive <primary datetime field>s of the <interval value expression> or <interval term> are subtracted from the corresponding fields of the <datetime value expression> or <datetime term>.

b) If, after the preceding step, any <primary datetime field> of the result is outside the permissible range of values for the field or the result is invalid based on the natural rules for dates and times, then an exception condition is raised: *data exception — datetime field overflow*.
NOTE 88 — For the permissible range of values for <primary datetime field>s, see Table 11, ''Valid values for datetime fields".

6.29 <interval value expression>

1. Rationale: Provide missing specification for leading field precision.

Replace Syntax Rule 2) with:

2) Case:

a) If the <interval value expression> simply contains an <interval qualifier> *IQ*, then the declared type of the result is INTERVAL *IQ*.

b) If the <interval value expression> is an <interval term>, then the result of the <interval value expression> contains the same interval fields as the <interval primary>. If the <interval primary> contains a seconds field, then the result's fractional seconds precision is the same as the <interval primary>'s fractional seconds precision. The result's <interval leading field precision> is implementation-defined, but shall not be smaller than the <interval leading field precision> of the <interval primary>.

c) If <interval term 1> is specified, then the result contains every interval field that is contained in the result of either <interval value expression 1> or <interval term 1>, and, if both contain a seconds field, then the fractional seconds precision of the result is the greater of the two fractional seconds precisions. The <interval leading field precision> is implementation-defined, but shall be sufficient to represent all interval values with the interval fields and <interval leading field precision> of <interval value expression 1> as well as all interval values with the interval fields and <interval leading field precision> of <interval term 1>.

2. Rationale: Editorial.

Replace General Rule 3) with:

3) If *IP* is an <interval primary>, then

Case:

a) If *IP* immediately contains a <value expression primary> *VEP*, then the value of *IP* is the value of *VEP*.

b) If *IP* is an <interval value function> *IVF*, then the value of *IP* is the value of *IVF*.

6.30 <boolean value expression>

1. Rationale: Correct use of undefined non-terminal.

Replace Syntax Rule 1) and 2) with:

1) The declared type of a <nonparenthesized value expression primary> shall be boolean.

2) If NOT is specified in a <boolean test>, then let *BP* be the contained <boolean primary> and let *TV* be the contained <truth value>. The <boolean test> is equivalent to:

```
( NOT ( BP IS TV ) )
```

Replace Syntax Rule 3) b) with:

b) If *BVE* is a <parenthesized boolean value expression> and the immediately contained <boolean value expression> is a known-not-null condition for *X*, then *BVE* is a *known-not-null condition* for *X*.

b.1) If *BVE* is a <nonparenthesized value expression primary>, then *BVE* is not a known-not-null condition for *X*.

7.1 <row value constructor>

1. Rationale: Specify Syntax Rules for "contextually typed" terms.

Insert the following Syntax Rules:

4.1) A <contextually typed row value constructor element> immediately contained in a <contextually typed row value constructor> shall not be a <value expression> of the form "<left paren> <value expression> <right paren>"

NOTE 93.1 — This Rule removes a syntactic ambiguity. A <contextually typed row value constructor> of this form is permitted, but is parsed in the form "<left paren> <contextually typed row value constructor list> <right paren>".

4.2) A <contextually typed row value constructor element> immediately contained in a <contextually typed row value constructor> shall not be a <value expression> that is a <row value expression>.

NOTE 93.2 — This Rule removes a syntactic ambiguity, since otherwise a <contextually typed row value constructor> could be a <row value expression>, and a <row value expression> could be a <contextually typed row value constructor>.

4.3) Let *CTRVC* be the <contextually typed row value constructor>. The declared type of *CTRVC* is a row type described by a sequence of (<field name>, <data type>) pairs, corresponding in order to each <contextually typed row value constructor element> *X* simply contained in *CTRVC*. The data type is the declared type of *X* and the <field name> is implementation-dependent and not equivalent to the <column name> name of any column or field, other than itself, of a table referenced by any <table reference> contained in the SQL-statement.

4.4) The degree of a <contextually typed row value constructor> is the degree of its declared type.

2. *Rationale: remove an ambiguity caused by the fact that a <predicate> is a <boolean value expression>; also, make the predicates non-associative.*

Insert the following Syntax Rule:

4.5) A <row value constructor> simply contained in a <row value expression> simply contained in a <predicate> shall not be a <row value constructor element> that is a <predicate>.

3. *Rationale: remove an ambiguity caused by the implicit promotion of scalars to rows. The implicit promotion is only desired in two contexts: predicates and <table value constructor>s.*

Insert the following Syntax Rule:

4.6) A <row value constructor> that is not simply contained in a <predicate> or a <table value constructor> shall not be a <row value constructor element>.

4. *Rationale: Supply missing Conformance Rules for <contextually typed row value constructor>*

Insert the following Conformance Rules:

5) Without Feature F641, "Row and table constructors", a <contextually typed row value constructor> that is not simply contained in a <contextually typed table value constructor> shall not contain more than one <row value constructor element>.

6) Without Feature F641, "Row and table constructors", a <contextually typed row value constructor> shall not be a <row subquery>.

7.2 <row value expression>

1. *Rationale: correct some ambiguities*

In the Format replace the production for <row value special case> with:

```
<row value special case> ::=
     <nonparenthesized value expression primary>
```

2. *Rationale: correct some ambiguities*

Replace General Rule 1) with:

1) A <row value special case> specifies the row value denoted by the <nonparenthesized value expression primary>.

7.3 <table value constructor>

1. *Rationale: Correct the definition of possibly non-deterministic.*

Replace Syntax Rule 3) with:

3) A <table value constructor> or a <contextually typed table value constructor> is *possibly non-deterministic* if it generally contains a possibly non-deterministic <value expression>.

2. *Rationale: Editorial.*

Replace Syntax Rule 4) with:

4) Let *TVC* be some <table value constructor> consisting of *n* <row value expression>s or some <contextually typed table value constructor> consisting of *n* <contextually typed row value expression>s. Let RVE_i, 1 (one) ≤ *i* ≤ *n*, denote the *i*-th <row value expression> or the *i*-th <contextually typed row value expression>. The row type of *TVC* is determined by applying Subclause 9.3, ''Data types of results of aggregations'', to the row types RVE_i, 1 (one) ≤ *i* ≤ *n*.

3. *Rationale: Use the correct non-terminal symbols.*

Replace Conformance Rule 1) with:

1) Without Feature F641, "Row and table constructors", the <contextually typed row value expression list> of a <contextually typed table value constructor> shall contain exactly one <contextually typed row value constructor> *RVE*. *RVE* shall be of the form "(<contextually typed row value constructor element list>)".

4. *Rationale: Supply missing Conformance Rules for <contextually typed table value constructor>*

Insert the following Conformance Rules:

3) Without Feature F641, "Row and table constructors", the <contextually typed row value expression list> of a <contextually typed table value constructor> shall contain exactly one <contextually typed row value constructor> *RVE*. *RVE* shall be of the form "(<contextually typed row value constructor element list>)".

7.6 <table reference>

1. *Rationale: Use correct containment.* Treat a collection of a row type differently than a collection of any other data type. Fix the equivalence of *CDT* and *ELDT*, which actually is the equivalence of *TR* and *ELDT*.

Replace Syntax Rules 1) and 2) with:

1) If a <table reference> *TR* simply contains a <collection derived table> *CDT*, then let *C* be the <collection value expression> simply contained in *CDT*, let *CN* be the <correlation name> simply contained in *TR*, and let *TEMP* be an <identifier> that is not equivalent to *CN* nor to any other <identifier> contained in *TR*. Let *ET* be the element type of the declared type of *C*.

a) Let *N1* and *N2* be two <column name>s that are not equivalent to one another nor to *CN*, *TEMP*, or any other <identifier> contained in *TR*.

b) Let *RECQP* be:

```
WITH RECURSIVE TEMP ( N1, N2 ) AS
  ( SELECT C[1] AS N1, 1 AS N2
      FROM ( VALUES( 1 ) ) AS CN
     WHERE 0 < CARDINALITY( C )
    UNION
    SELECT C[N2+1] AS N1, N2+1 AS N2
      FROM TEMP
     WHERE N2 < CARDINALITY( C )
  )
```

c) Case:

i) If *TR* specifies a <derived column list> *DCL*, then

1) Case:

A) If *CDT* specifies WITH ORDINALITY, then

Case:

I) If *ET* is a row type, then let *DET* be the degree of *ET*. *DCL* shall contain *DET* + 1 (one) <column name>s.

II) Otherwise, *DCL* shall contain 2 <column name>s.

B) Otherwise,

Case:

I) If *ET* is a row type, then let *DET* be the degree of *ET*. *DCL* shall contain *DET* <column name>s.

II) Otherwise, *DCL* shall contain one <column name>.

2) Let *PDCLP* be:

(*DCL*)

ii) Otherwise,

Case:

1) If *ET* is a row type, then

A) Let *DET* be the degree of *ET*.

B) For i, $1 \le i \le DET$, let FN_i be the name of the i-th field in *ET*.

C) Case:

I) If *CDT* specifies WITH ORDINALITY, then let *PDCLP* be

(FN_1, FN_2, ..., FN_{DET}, *N2*)

II) Otherwise, let *PDCLP* be

(FN_1, FN_2, ..., FN_{DET})

2) Otherwise, let *PDCLP* be a zero-length string.

d) Case:

i) If *CDT* specifies WITH ORDINALITY, then

Case:

1) If *ET* s a row type, then let *ELDT* be:

```
LATERAL ( RECQP SELECT N1.*, N2 FROM TEMP )
AS CN PDCLP
```

2) Otherwise, let *ELDT* be:

```
LATERAL ( RECQP SELECT * FROM TEMP ) AS CN PDCLP
```

ii) Otherwise,

Case:

1) If *ET* is a row type, then let *ELDT* be:

```
LATERAL ( RECQP SELECT N1.* FROM TEMP ) AS CN PDCLP
```

2) Otherwise, let *ELDT* be:

```
LATERAL ( RECQP SELECT N1 FROM TEMP AS ) CN PDCLP
```

e) *TR* is equivalent to the <table primary> *ELDT*.

2) A <correlation name> simply contained in a <table reference> *TR* is *exposed* by *TR*. A <table or query name> simply contained in a <table reference> *TR* is *exposed* by *TR* if and only if *TR* does not specify a <correlation name>.

2. *Rationale: The scope of an exposed table name or correlation name must include the <order by clause> of a simple table query. Use correct containment.*

Replace Syntax Rule 3) with:

3) Case:

a) If a <table reference> *TR* is contained in a <from clause> *FC* with no intervening <query expression>, then the *scope clause SC* of *TR* is the <select statement: single row> or innermost <query specification> that contains *FC*. The scope of the exposed <correlation name> or exposed <table or query name> of *TR* is the <select list>, <where clause>, <group by clause>, and <having clause> of *SC*, together with every <lateral derived table> that is simply contained in *FC* and is

preceded by *TR*, and every <collection derived table> that is simply contained in *FC* and is preceded by *TR*, and the <join condition> of all <joined table>s contained in *SC* that contain *TR*. If *SC* is the <query specification> that is the <query expression body> of a simple table query *STQ*, then the scope of the exposed <correlation name> or exposed <table or query name> also includes the <order by clause> of *STQ*.

b) Otherwise, the *scope clause SC* of *TR* is the outermost <joined table> that contains *TR* with no intervening <query expression>. The scope of the exposed <correlation name> or exposed <table or query name> of *TR* is the <join condition> of *SC* and of all <joined table>s contained in *SC* that contain *TR*.

3. *Rationale: supply correct rules for equivalency of <table name>s, <query name>s and <correlation name>s.*

Replace Syntax Rules 4) and 5) with:

4) A <table or query name> that is a <table name> that is exposed by a <table reference> *TR* shall not be equivalent to any other <table or query name> that is a <table name> that is exposed by a <table reference> with the same scope clause as *TR*.

4.1) A <table or query name> that is a <query name> that is exposed by a <table reference> *TR* shall not be equivalent to the <qualified identifier> of any <table or query name> that is a <table name> that is exposed by a <table reference> with the same scope clause as *TR* , and shall not be equivalent to any other <table or query name> that is a <query name> that is exposed by a <table reference> with the same scope clause as *TR*.

5) A <correlation name> that is exposed by a <table reference> *TR* shall not be equivalent to any other <correlation name> that is exposed by a <table reference> with the same scope clause as *TR* and shall not be equivalent to the <qualified identifier> of any <table or query name> that is a <table name> that is exposed by a <table reference> with the same scope clause as *TR*, and shall not be equivalent to any <table or query name> that is a <query name> that is exposed by a <table reference> with the same scope clause as *TR*.

4. *Rationale: Use correct containment.*

Replace Syntax Rules 6) and 7) with:

6) A <table or query name> simply contained in a <table reference> *TR* has a scope clause and scope defined by that <table reference> if and only if the <table or query name> is exposed by *TR*.

7) If *TR* simply contains <only spec> *OS* and the table identified by the <table or query name> *TN* is not a typed table, then *OS* is equivalent to *TN*.

Replace Syntax Rule 11) with:

11) Case:

a) If no <derived column list> is specified, then the row type *RT* of the <table reference> is the row type of its simply contained <table or query name>, <derived table>, <lateral derived table>, or <joined table>.

b) Otherwise, the row type *RT* of the <table reference> is described by a sequence of (<field name>, <data type>) pairs, where the <field name> in the *i*-th pair is the *i*-th <column name> in the <derived column list> and the <data type> in the *i*-th pair is the declared type of the *i*-th column of the <derived table>, <joined table>, or <lateral derived table>, of the table identified by the <table or query name> simply contained in the <table reference>.

Replace Syntax Rule 12) and 13) with:

12) A <derived table> or <lateral derived table> is an *updatable derived table* if and only if the <query expression> simply contained in *TR* is updatable.

13) A <derived table> or <lateral derived table> is an *insertable-into derived table* if and only if the <query expression> simply contained in *TR* is insertable-into.

Replace Syntax Rule 15) c) with:

15) c) If *TR* immediately contains a <derived table> or <lateral derived table>, then every updatable column of the table identified by the <query expression> simply contained in <derived table> or <lateral derived table> is called an *updatable column* of *TR*.

Replace Syntax Rule 16) with:

16) If the <table or query name> simply contained in <table reference> is not a query name in scope, then let *T* be the table identified by the <table name> immediately contained in <table or query name>. If the <table reference> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*. If the <table reference> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*, or *S* shall contain a <schema element> that creates the descriptor of *T*.
NOTE 94 — "*query name in scope*" is defined in Subclause 7.12, "<query expression>".

5. *Rationale: Define possibly non-deterministic <table reference> and <table primary>.*

Insert the following Syntax Rules:

17) A <table name> is *possibly non-deterministic* if the table identified by the <table name> is a viewed table, and the original <query expression> in the view descriptor identified by the <table name> is possibly non-deterministic.

18) A <query name> is *possibly non-deterministic* if the <query expression> identified by the <query name> is possibly non-deterministic.

19) A <derived table> or <lateral derived table> is *possibly non-deterministic* if the simply contained <query expression> is possibly non-deterministic.

20) A <table primary> is *possibly non-deterministic* if the simply contained <table name>, <query name>, <derived table>, <lateral derived table>, or <joined table> is possibly non-deterministic.

21) A <table reference> is *possibly non-deterministic* if the simply contained <table primary> or <joined table> is possibly non-deterministic.

6. *Rationale: Use correct containment and correct the <only spec> and untyped tables. Also, there is no <row value expression> immediately contained in a <set clause>.*

Replace Access Rule 1) with:

1) If <table reference> simply contains a <table or query name> that is a <table name>, then:

a) Let *T* be the table identified by the <table name> immediately contained in the <table or query name> simply contained in <table reference>.

b) If *T* is a base table or a viewed table and the <table reference> is contained in any of:

— A <query expression> simply contained in a <cursor specification>, a <view definition>, or an <insert statement>.

— A <table expression> or <select list> immediately contained in a <select statement: single row>.

— A <search condition> immediately contained in a <delete statement: searched> or an <update statement: searched>.

— A <value expression> simply contained in an <update source>.

then

Case:

i) If <table reference> is contained in an <SQL schema statement> then, the applicable privileges of the <authorization identifier> that owns the containing schema shall include SELECT on at least one column of *T*.

ii) Otherwise, the current privileges shall include SELECT on at least one column of *T*.

NOTE 95 — "applicable privileges" and "current privileges" are defined in Subclause 10.5, "<privileges>".

c) If the <table reference> is contained in a <query expression> simply contained in a <view definition>, then the applicable privileges of the <authorization identifier> that owns the view shall include SELECT for at least one column of *T*.

d) If *TR* simply contains <only spec> and *TR* identifies a typed table, then

Case:

i) If <table reference> is contained in a <schema definition>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include SELECT WITH HIERARCHY OPTION on at least one supertable of *T*.

ii) Otherwise, the current privileges shall include SELECT WITH HIERARCHY OPTION on at least one supertable of *T*.

NOTE 96 — "applicable privileges" and *current privileges* are defined in Subclause 10.5, "<privileges>".

7. *Rationale: Use correct containment.*

Replace General Rule 1) with:

1) A <correlation name> or exposed <table or query name> simply contained in a <table reference> defines that <correlation name> or <table or query name> to be an identifier of the table identified by the <table or query name> or <derived table> or <lateral derived table> of that <table reference>.

8. *Rationale: The scope of a <table reference> must include <window clause>*

Replace General Rule 3) with:

3) If a <derived table> or <lateral derived table> *LDT* simply containing <query expression> *QE* is specified, then the result of *LDT* is the result of *QE*.

7.7 <joined table>

1. *Rationale: Outer join, combined with LATERAL doesn't always work*

Insert the following Syntax Rule:

1.1) TR_2 shall not contain a <lateral derived table> containing an outer reference that references TR_1.

2. *Rationale: The second operand of <joined table> may be a <table primary>.*

Replace Syntax Rule 2) with:

2) Let TR_1 be the first <table reference> and let TR_2 be the <table reference> or <table primary> that is the second operand of the <joined table>. Let T_1 and T_2 be the tables identified by TR_1 and TR_2, respectively. Let *TA* and *TB* be the correlation names of TR_1 and TR_2, respectively. Let *CP* be:

SELECT * FROM TR_1, TR_2

3. *Rationale: Correct the definition of possibly non-deterministic.*

Replace Syntax Rule 3) with:

3) A <joined table> is *possibly non-deterministic* if at least one of the following conditions is true:

a) either TR_1 or TR_2 is possibly non-deterministic, or

b) a <join condition> that generally contains a possibly non-deterministic <value expression>, possibly non-deterministic <query specification> or possibly non-deterministic <query expression> is specified, or

c) NATURAL is specified, or a <join specification> immediately containing a <named columns join> is specified, and there is a common column name *CCN* such that the declared types of the two corresponding join columns identified by *CCN* have corresponding constituents such that one constituent is datetime with time zone and the other is datetime without time zone.

.

4. *Rationale: NATURAL joins should be subject to the same syntactic restrictions as <named column join>s. Provide consistent Syntax Rules for comparison operations.*

Replace Syntax Rule 7) c) with:

7) c) Let C_1 and C_2 be a pair of corresponding join columns contained in T_1 and T_2, respectively. C_1 and C_2 shall be comparable, and the declared type of C_1 or C_2 shall not be LOB-ordered or array_ordered.

5. *Rationale: Named column joins implicitly perform <comparison predicate>s, and should be subject to the same Conformance Rules. Provide consistent Conformance Rules for comparison operations.*

Insert the following Conformance Rule:

4.1) Without Feature S024, "Enhanced structured types", if NATURAL or <named column join> is specified, and if C is a corresponding join column, then the declared type of C shall not be an ST-ordered type.

7.8 <where clause>

1. *Rationale: Clarification.*

Replace Syntax Rule 2) with:

2) If a <value expression> directly contained in the <search condition> is a <set function specification>, then the <where clause> shall be contained in a <having clause> or <select list>, the <set function specification> shall contain a column reference, and every column reference contained in the <set function specification> shall be an outer reference .

7.9 <group by clause>

1. *Rationale: Disambiguate BNF non-terminal names; provide a unified treatment of CUBE and ROLLUP.*

In the Format, replace the production for <group by clause> with:

```
<group by clause> ::= GROUP BY <grouping element list>
```

Insert the following production:

```
<grouping element list> ::=
    <grouping element> [ {<comma> <grouping element>}...]
```

In the Format, replace the production for <grouping specification> with:

```
<grouping element> ::=
        <ordinary grouping set>
      | <rollup list>
      | <cube list>
      | <grouping sets specification>
      | <grand total>
```

In the Format, replace the production for <grouping sets list> with:

```
<grouping sets specification> ::=
     GROUPING SETS <left paren> <grouping set list> <right paren>
```

Replace the production for <grouping set> with:

```
<grouping set> ::=
       <ordinary grouping set>
     | <rollup list>
     | <cube list>
     | <grouping sets specification>
     | <grand total>
```

2. *Rationale: Extend the restrictions on the use of columns with types based on LOB and array types. Provide consistent Syntax Rules for comparison operations.*

Delete Syntax Rule 2).

3. *Rationale: Provide a correct, unified treatment of CUBE and ROLLUP.*

Replace Syntax Rule 3) with:

3) Let *QS* be the <query specification> that simply contains the <group by clause>, and let *SL*, *FC*, *WC*, *GBC* and *HC* be the <select list>, the <from clause>, the <where clause> if any, the <group by clause> and the <having clause> if any, respectively, that are simply contained in *QS*. Let *QSSQ* be the explicit or implicit <set quantifier> immediately contained in *QS*.

Delete Syntax Rule 4).

4. *Rationale: Extend the restrictions on the use of columns with types based on LOB and array types. Provide consistent Syntax Rules for comparison operations.*

Replace Syntax Rule 5) with:

5) The declared type of a grouping column shall not be LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

5. *Rationale: Provide a correct, unified treatment of CUBE and ROLLUP*

Insert the following Syntax Rules:

6.1) A <grouping set list> shall not contain a <grouping sets specification>.

6.2) If a <group by clause> simply contains a <grouping sets specification> *GSS*, then *GSS* shall be the only <grouping element> simply contained in the <group by clause>.

6.3) Let *SL1* be obtained from *SL* by replacing every <asterisk> and <asterisked identifier chain> using the syntactic transformations in the Syntax Rules of Subclause 7.12 “<query specification>”.

6.4) A <group by clause> is *primitive* if it does not contain a <rollup list>, <cube list>, <grouping sets specification>, or <grouping column reference list>, and does not contain both a <grouping column reference> and an <empty grouping set>.

6.5) A <group by clause> is *simple* if it does not contain a <rollup list>, <cube list> or <grouping sets specification>.

6.6) If *GBC* is a simple <group by clause> that is not primitive, then *GBC* is transformed into a primitive <group by clause> as follows:

a) Let *NSGB* be the number of <grouping column reference>s contained in *GBC*.

b) Case:

i) If *NSGB* is 0 (zero), then *GBC* is replaced by

```
GROUP BY ()
```

ii) Otherwise:

1) Let $SGCR_1$, ... $SGCR_{NSGB}$ be an enumeration of the <grouping column reference>s contained in *GBC*.

2) *GBC* is replaced by

GROUP BY $SGCR_1$, ... , $SGCR_{NSGB}$

NOTE 101.1 — that is, a simple <group by clause> that is not primitive may be transformed into a primitive <group by clause> by deleting all parentheses, and deleting extra <comma>s as necessary for correct syntax. If there are no grouping columns at all (for example, GROUP BY (), ()) this is transformed to the canonical form GROUP BY ().

6.7) If *GBC* is a primitive <group by clause>, then let *SLNEW* and *HCNEW* be obtained from *SL1* and *HC*, respectively, by replacing every <grouping operation> by the exact numeric literal 0 (zero). *QS* is equivalent to

```
SELECT QSSQ SLNEW FC WC GBC HCNEW
```

Replace Syntax Rules 7), 8), 9), 10) and 11) with:

7) If *RL* is a <rollup list>, then let GCR_i range over the *n* <grouping column reference>s contained in *RL*. *RL* is equivalent to

GROUPING SETS (
(GCR_1, GCR_2 , ..., GCR_n),
(GCR_1 , GCR_2 , ..., GCR_{n-1}),
(GCR_1 , GCR_2 , ..., GCR_{n-2}),
...
(GCR_1),
()
)

NOTE 102 — The result of the transform is to replace *RL* with a <grouping sets specification> that contains a <grouping set> for every initial sublist of the <grouping column reference list> of the <rollup list> , obtained by

dropping elements from the right, one by one , and regarding <empty grouping set> as the shortest such initial sublist.

8) If *CL* is a <cube list>, then let GCR_i range over the *n* <grouping column reference>s contained in *CL*. *CL* is transformed as follows:

a) Let $M = 2^n - 1$.

b) For each *i* between 0 (zero) and *M*:

i) Let BSL_i be the <bit string literal> containing *n* <bit>s whose binary value is *i*.

ii) For each *j* between 1 (one) and *n*, let $B_{i,j}$ be the *j*-th <bit>, counting from left to right, in BSL_i.

iii) For each *j* between 1 (one) and *n*, let $GSLCR_{i,j}$ be

Case:

1) If $B_{i,j}$ is 0 (zero), then the zero-length string.

2) If $B_{i,j}$ is 1 (one), and $B_{i,k}$ is 0 (zero) for all $k < j$, then GCR_j.

3) Otherwise, <comma> GCR_j.

iv) Let GSL_i be the <ordinary grouping set>

($GSLCR_{i,1}$ $GSLCR_{i,2}$ ... $GSLCR_{i,n}$)

c) *CL* is equivalent to:

GROUPING SETS (GSL_M , GSL_{M-1} , . . . GSL_0)

NOTE 103 — The result of the transform is to replace *CL* with a <grouping sets specification> that contains a <grouping set> for all possible subsets of the set of grouping columns in the <grouping column reference list> of the <cube list>, including <empty grouping set> as the empty subset with no grouping columns. For example, CUBE (A, B, C) is equivalent to:

```
GROUPING SETS (/* BSL_i */
( A, B, C),     /* 111 */
( A, B   ),     /* 110 */
( A,    C),     /* 101 */
( A      ),     /* 100 */
(    B, C),     /* 011 */
(    B   ),     /* 010 */
(       C),     /* 001 */
(        )      /* 000 */
)
```

9) If <grouping sets specification> *GSSA* simply contains another <grouping sets specification> *GSSB*, then *GSSA* is transformed as follows:

a) Let *NA* be the number of <grouping set>s simply contained in *GSSA*, and let *NB* be the number of <grouping set>s simply contained in *GSSB*.

b) Let GSA_i be an enumeration of the <grouping set>s simply contained in *GSSA*, for *i* between 1 (one) and *NA*.

c) Let GSB_i be an enumeration of the <grouping set>s simply contained in *GSSB*, for *i* between 1 (one) and *NB*.

d) Let *k* be the index such that $GSSB = GSA_k$.

e) *GSSA* is equivalent to:

```
GROUPING SETS
  ( GSA1 , GSA2 , ... GSAk-1 ,
    GSB1 , ... , GSBNB,
    GSAk+1 , ..., GSANA
  )
```

NOTE 103.1 — Thus the nested <grouping sets specification> is removed by simply "promoting" each of its <grouping set>s to be a <grouping set> of the encompassing <grouping sets specification>.

10) If *CGB* is a <group by clause> that is not simple, then *CGB* is transformed as follows:

a) Previous Syntax Rules are applied repeatedly to eliminate any <grouping set specification> that is nested in another <grouping set specification>, as well as any <rollup list> and any <cube list>.

NOTE 103.2 — As a result, *CGB* is a list of two or more <grouping set>s, each of which is an <ordinary grouping set>, an <empty grouping set> or a <grouping sets specification> that contains only <ordinary grouping set>s and <empty grouping set>s. There are no remaining <rollup list>s, <cube list>s, or nested <grouping sets specification>s.

b) Any <grouping element> *GS* that is an <ordinary grouping set> or an <empty grouping set> is replaced by the <grouping sets specification>:

```
GROUPING SETS ( GS )
```

NOTE 103.3 — As a result, *CGB* s a list of two or more <grouping sets specification>s.

c) Let *GSSX* and *GSSY* be the first two <grouping sets specification>s in *CGB*. *CGB* is transformed by replacing *GSSX* <comma> *GSSY* as follows:

i) Let *NX* be the number of <grouping set>s in *GSSX* and let *NY* be the number of <grouping set>s in *GSSY*.

ii) Let GSX_i for *i* between 1 (one) and *NX* be the <grouping set>s contained in *GSSX*, and let GSY_i for *i* between 1 (one) and *NY* be the <grouping set>s contained in *GSSY*.

iii) Let *MX(i)* be the number of <grouping column reference>s in GSX_i, and let *MY(i)* be the number of <grouping column reference>s in GSY_i.

NOTE 103.4 — If GSX_i is <empty grouping set>, then *MX(i)* is 0 (zero); and similarly for GSY_i.

iv) Let $GCRX_{i,j}$ for *j* between 1 (one) and *MX(i)* be the <grouping column reference>s contained in GSX_i, and let $GCRY_{i,j}$ for *j* between 1(one) and *MY(i)* be the <grouping column reference>s contained in GSY_i.

NOTE 103.5 — If GSX_i is <empty grouping set>, then there are no $GCRX_{i,j}$; and similarly for GSY_i.

v) For each *a* between 1 (one) and *NX* and each *b* between 1 (one) and *NY*, let $GST_{a,b}$ be

($GCRX_{a,1}$, ... , $GCRX_{a,MX(a)}$, $GCRY_{b,1}$, ... , $GCRY_{b,MY(b)}$)

that is, an <ordinary grouping set> consisting of $GCRA_{a,j}$ for all *j* between 1 (one) and *MX(a)* followed by $GCRY_{b,j}$ for all *j* between 1 (one) and *MY(b)*.

vi) *CGB* is transformed by replacing *GSSX* <comma> *GSSY* with:

```
GROUPING SETS
  ( GST1,1 , ... GST1,NY ,
    GST2,1 , ... GST2,NY ,
    . . .
    GSTNX,1 , ... GSTNX,NY
  )
```

NOTE 103.6 — Thus each <ordinary grouping set> in *GSSA* is "concatenated" with each <ordinary grouping set> in *GSSB*. For example,

```
GROUP BY GROUPING SETS ( ( A, B ), ( C ) ),
GROUPING SETS ( ( X, Y ), ( ) )
```

is transformed to.

```
GROUP BY GROUPING SETS (
  ( A, B, X, Y ),
  ( A, B ),
  ( C, X, Y ), ( C ) )
```

d) The previous subrule of this Syntax Rule is applied repeatedly until *CGB* consists of a single <grouping sets specification>.

11) If <grouping specification> consists of a single <grouping sets specification> *GSS* that contains only <ordinary grouping set>s or <grand total>, then:

a) Let *m* be the number of <grouping set>s contained in *GSS*.

b) Let G_{si}, 1 <= *i* <= *m*, range over the <grouping set>s contained in *GSS*.

c) Let *p* be the number of distinct <column reference>s that are contained in *GSS*.

d) Let *PC* be an ordered list of these <column reference>s ordered according to their left-to-right occurrence in the list.

e) Let PC_k, 1 <= *k* <= *p*, be the *k*-th <column reference> in *PC*.

f) Let $DTPC_k$ be the declared type of the column identified by PC_k.

g) Let $CNPC_k$ be the column name of the column identified by PC_k.

h) For each GS_i:

i) If GS_i is a <grand total>, then let *n(i)* be 0 (zero). If GS_i is a <grouping column reference>, then let *n(i)* be 1 (one). Otherwise, let *n(i)* be the number of <grouping column reference>s contained in the <grouping column reference list>.

ii) Let $GCR_{i,j}$, 1 <= *j* <= *n(i)*, range over the <grouping column reference>s contained in GS_i.

iii) Case:

1) If GS_i is an <ordinary grouping set>, then

A) Transform *SL1* to obtain *SL2*, and transform *HC* to obtain *HC2*, as follows:

For every PC_k:

If there is no *j* such that $PC_k = GCR_{i,j}$, then make the following replacements in *SL1* and *HC*:

I) Replace each <grouping operation> in *SL1* and *HC* that contains a <column reference> that references PC_k by the <literal> 1 (one).

II) Replace each <derived column> in *SL1* that is a <column reference> that references PC_k by:

```
CAST ( NULL AS DTPC_k ) AS CNPC_k
```

III) Replace each <column reference> in *SL1* and *HC* that references PC_k and that is not an entire <derived column> by:

```
CAST ( NULL AS DTPC_k )
```

B) Transform *SL2* to obtain *SLNEW*, and transform *HC2* to obtain *HCNEW* by replacing each <grouping operation> that remains in *SL2* and *HC2* by the <literal> 0 (zero).

NOTE 103.7 — Thus the value of a <grouping operation> is 0 (zero) if the grouping column referenced by the <grouping operation> is among the $GCR_{i,j}$ and 1(one) if it is not.

C) Let $GSSQL_i$ be:

```
SELECT SLNEW FC WC
GROUP BY GCR_i,1 , ..., GCR_i,n(i)
HCNEW
```

2) If GS_i is a <grand total>, then

A) Transform *SL1* to obtain *SLNEW*, and transform *HC* to obtain *HCNEW*, as follows:

For every *k*, 1 <= *k* <= *p*:

I) Replace each <grouping operation> that contains a <column reference> that references PC_k by the <literal> 1 (one).

II) Replace each <derived column> in *SL1* that is a <column reference> that references PC_k by:

CAST (NULL AS *DTPC*$_k$) AS *CNPC*$_k$

III) Replace each <column reference> in *SL1* and *HC* that references *PC*$_k$ and that is not an entire <derived column> by:

CAST (NULL AS *DTPC*$_k$) AS *CNPC*$_k$

B) Let *GSSQL*$_i$ be:

```
SELECT SLNEW FC WC
GROUP BY ( )
HCNEW
```

i) *QS* is equivalent to:

```
GSSQL1
UNION QSSQ
GSSQL2
UNION QSSQ
.
.
.
UNION QSSQ
GSSQLm
```

Delete Syntax Rule 12).

6. Rationale: clarify the semantics of column references.

Replace General Rule 3) with:

3) When a <search condition> or <value expression> is applied to a group, then a reference *CR* to a column that is functionally dependent on the grouping columns is understood as follows.

Case:

a) If *CR* is a group-invariant column reference, then it is a reference to the common value in that column of the rows in that group. If the most specific type of the column is character, datetime with time zone, or a user-defined type, the value is an implementation-dependent value that is not distinct from the value of the column in each row of the group.

b) Otherwise, *CR* is a within-group-varying column reference, and as such, it is a reference to the value of the column in each row of a given group determined by the grouping columns, to be used to construct the argument source of a <set function specification>.

7. Rationale: Adapting the Conformance Rules of <group by clause> to the Feature name and broader scope of Feature T431, "Extended grouping capabilities".

Replace Conformance Rule 1) with:

1) Without Feature T431, "Extended grouping capabilities", conforming SQL language shall not specify ROLLUP, CUBE, GROUPING SETS, or <grand total>.

8. Rationale: Provide consistent Conformance Rules for comparison operations.

Replace Conformance Rule 3) with:

3) Without Feature S024, "Enhanced structured types", a grouping column shall not be of an ST-ordered declared type.

9. Rationale: Adapting the Conformance Rules of <group by clause> to the Feature name and broader scope of Feature T431, "Extended grouping capabilities".

Insert the following Conformance Rule:

3.1) Without Feature T431, "Extended grouping capabilities", an <ordinary grouping set> shall be a <grouping column reference>.

7.10 <having clause>

1. Rationale: Supply missing application of functional dependencies to <having clause>.

Insert the following Conformance Rules:

1) Without Feature T301, "Functional dependencies", each column reference directly contained in the <search condition> shall be one of the following:

   a) an unambiguous reference to a grouping column of *T*, or

   b) an outer reference.

2) Without Feature T301, "Functional dependencies", each column reference contained in a <subquery> in the <search condition> that references a column of *T* shall be one of the following:

   a) an unambiguous reference to a grouping column of *T*, or

   b) contained in a <set function specification>.

7.11 <query specification>

1. Rationale: Disallow DISTINCT on columns with types based on LOB and array types. Provide consistent Syntax Rules for comparison operations.

Replace Syntax Rule 5) with:

5) If a <set quantifier> DISTINCT is specified, then no column of *T* shall have a declared type that is LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

2. Rationale: Permit outer references in a <select list>.

Delete Syntax Rule 10).

3. *Rationale: Correct the definition of possibly non-deterministic.*

Replace Syntax Rule 11) with:

11) A <query specification> is *possibly non-deterministic* if any of the following conditions are true:

a) The <set quantifier> DISTINCT is specified and one of the columns of *T* has a data type of character string, user-defined type, TIME WITH TIME ZONE, or TIMESTAMP WITH TIME ZONE.

b) The <query specification> generally contains a <value expression>, <query specification> or <query expression> that is possibly non-deterministic.

c) The <query specification> directly contains a <having clause> that is possibly non-deterministic.

d) The <select list> contains a reference to a column *C* of *T* that has a data type of character string, user-defined type, TIME WITH TIME ZONE, or TIMESTAMP WITH TIME ZONE, and the functional dependency $G \rightarrow C$, where *G* is the set consisting of the grouping columns of *T*, holds in *T*.

4. *Rationale: Clarify which <value expression>s are meant.* Clarify the semantics of column reference.

Replace Syntax Rule 13) with:

13) If *T* is a grouped table, then let *G* be the set consisting of every column referenced by a <column reference> contained in the <group by clause> immediately contained in <table expression>. In each <value expression> contained in the <select list>, each <column reference> that references a column of *T* shall reference some column *C* that is functionally dependent on *G* or shall be contained in an aggregated argument of a <set function specification> whose aggregation query is *QS*.

5. *Rationale: Clarify when GROUP BY ( ) is implicit.*

Replace Syntax Rule 15) with:

15) If <table expression> does not immediately contain a <group by clause> and <table expression> is simply contained in a <query expression> that is the aggregation query of some <set function specification>, then GROUP BY ( ) is implicit.
NOTE 118.1 — The term *aggregation query* is defined in Subclause 6.16, "<set function specification>".

6. *Rationale: Certain predicates may be a source of a null even if all operands are known not null. Imprecise wording and bad grammar. A column cannot contain syntax elements.*

Replace Syntax Rule 16) with:

16) A column of *TQS* is *known not null* if and only if at least one of the following conditions applies.

a) It is not defined by a <derived column> containing any of the following:

i) A column reference for a column *C* that is possibly nullable.

ii) An <indicator parameter>.

iii) An SQL parameter.

iv) A <routine invocation>, <method reference>, or <method invocation> whose subject routine is an SQL-invoked routine that either is an SQL routine or is an external routine that specifies or implies PARAMETER STYLE SQL.

v) A <subquery>.

vi) CAST (NULL AS *X*) (where *X* represents a <data type> or a <domain name>).

vii) CURRENT_USER, CURRENT_ROLE, or SYSTEM_USER.

viii) A <set function specification> that does not contain COUNT.

ix) A <case expression>.

x) A <field reference>.

xi) An <element reference>.

xii) A <dereference operation>.

xiii) A <reference resolution>.

xiv) A <comparison predicate>, <between predicate>, <in predicate> or <quantified comparison predicate> *P* such that the declared type of a field of a <row value expression> that is simply contained in *P* is a row type, a user-defined type, or an array type.

b) An implementation-defined rule by which the SQL-implementation can correctly deduce that the value of the column cannot be null.

7. *Rationale: Clarify which <value expression>s are meant.*

Replace Conformance Rule 3) with:

3) Without Feature T301, "Functional dependencies", if *T* is a grouped table, then in each <value expression> contained in the <select list>, each <column reference> that references a column of *T* shall reference a grouping column or be specified in a <set function specification>.

8. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Replace Conformance Rule 4) with:

4) Without Feature S024, "Enhanced structured types", if any column in the result of a <query specification> is of an ST-ordered declared type, then DISTINCT shall not be specified or implied.

7.12 <query expression>

1. Rationale: Define an undefined term.

Insert the following Syntax Rule:

0.1) Let *QE* be the <query expression>.

2. Rationale: Problems with misuse of syntactic containment.

Replace Syntax Rule 1) a) with:

1) a) If a <with clause> *WC* immediately contains RECURSIVE, then *WC*, its <with list> and its <with list element>s are said to be *potentially recursive*. Otherwise they are said to be *non-recursive*.

Replace Syntax Rule 1) f) with:

1) f) A query name dependency graph *QNDG* of a potentially recursive <with list> *WL* is a directed graph such that, for *i* ranging from 1 (one) to the number of <query name>s simply contained in *WL*:

i) Each node represents a <query name> WQN_i immediately contained in a <with list element> WLE_i of *WL*.

ii) Each arc from a node WQN_i to a node WQN_j represents the fact that WQN_j is referenced by a <query name> contained in the <query expression> immediately contained in WLE_i. WQN_i is said to *depend immediately* on WQN_j.

3. Rationale: Correct spelling of a variable symbol.

Replace Syntax Rule 1) g) i) 1) with:

1) g) i) 1) If *QNDG* contains an arc from WQN_i to itself, then WLE_i, WQN_i, and WQT_i are said to be *recursive*. WQN_i is said to belong to the *stratum* of WQE_i.

Replace Syntax Rule 1) g) i) 2) with:

1) g) i) 2) If *QNDG* contains a cycle comprising WQN_i, ..., WQN_k, with $k \neq i$, then it is said that WQN_i, ..., WQN_k are *recursive* and *mutually recursive* to each other, WQT_i, ..., WQT_k are *recursive* and *mutually recursive* to each other, and WLE_i, ..., WLE_k are *recursive* and *mutually recursive* to each other.

For each *j* ranging from *i* to *k*, WQN_j belongs to the stratum of WQE_i,..., and WQE_k.

4. Rationale: Problems with misuse of syntactic containment.

Replace Syntax Rule 1) g) i) 3) B) with:

1) g) i) 3) B) WQE_j has one operand that does not contain a <query name> referencing any of WQN_i, ..., WQN_k. This operand is said to be the *non-recursive operand* of WQE_j.

Replace Syntax Rule 1) g) i) 3) D) with:

1) g) i) 3) D) Let *CCCG* be the subgraph of *QNDG* that contains no nodes other than WQN_i, ..., WQN_k. For any anchor name WQN_j, remove the arcs to those query names WQN_l that are referenced by any <query name> contained in WQE_j. The remaining graph *SCCGP* shall not contain any cycle.

Replace Syntax Rule 1) g) ii) 2) with:

1) g) ii) 2) Otherwise, let WQE_i contain any two <query name>s referencing WQN_k and WQN_l, both of which belong to the stratum of WQE_i.

Replace Syntax Rule 1) g) ii) 2) A) with:

1) g) ii) 2) A) WLE_i is linearly recursive if each of the following conditions is satisfied:

I) WQE_i does not contain a <table reference list> that contains <query name>s referencing both WQN_k and WQN_l.

II) WQE_i does not contain a <joined table> such that *TR1* and *TR2* are the first and second <table reference>s, respectively, and *TR1* and *TR2* contain <query name>s referencing WQN_k and WQN_l, respectively, except for union join.

III) WQE_i does not contain a <table expression> that immediately contains a <from clause> that contains WQN_k, and immediately contains a <where clause> containing a <subquery> that contains a <query name> referencing WQN_l.

Replace Syntax Rule 1) g) iii) with:

1) g) iii) For each WLE_i, for *i* ranging from 1 (one) to *n*, and for each WQN_j that belongs to the stratum of WQE_i:

1) WQE_i shall not contain a <non-join query expression> that contains a <query name> referencing WQN_j and immediately contains EXCEPT where the right operand of EXCEPT contains WQN_j.

2) WQE_i shall not contain a <routine invocation> with an <SQL argument list> that contains one or more <SQL argument>s that immediately contain a <value expression> that contains a <query name> referencing WQN_j.

3) WQE_i shall not contain a <table subquery> *TSQ* that contains a <query name> referencing WQN_j, unless *TSQ* is immediately contained in a <table reference> that is immediately contained in a <table expression> that is immediately contained in a <query specification> that is immediately contained in a <non-join query expression> that is WQE_i.

4) WQE_i shall not contain a <query specification> *QS* such that:

A) *QS* immediately contains a <table expression> *TE* that contains a <query name> referencing WQN_j, and

B) *QS* immediately contains a <select list> *SL* or *TE* immediately contains a <having clause> *HC* and *SL* or *TE* contain a <set function specification>.

5) WQE_i shall not contain a <non-join query expression> that contains a <query name> referencing WQN_j and immediately contains INTERSECT ALL or EXCEPT ALL.

6) WQE_i shall not contain a <qualified join> QJ in which:

A) QJ immediately contains a <join type> that specifies FULL and a <table reference> that contains a <query name> referencing WQN_j.

B) QJ immediately contains a <join type> that specifies LEFT and a <table reference> following the <join type> that contains a <query name> referencing WQN_j.

C) QJ immediately contains a <join type> that specifies RIGHT and a <table reference> preceding the <join type> that contains a <query name> referencing WQN_j.

7) WQE_i shall not contain a <natural join> QJ in which:

A) QJ immediately contains a <join type> that specifies FULL and a <table reference> or <table primary> that contains a <query name> referencing WQN_j.

B) QJ immediately contains a <join type> that specifies LEFT and a <table primary> following the <join type> that contains a <query name> referencing WQN_j.

C) QJ immediately contains a <join type> that specifies RIGHT and a <table reference> preceding the <join type> that contains a <query name> referencing WQN_j.

Replace Syntax Rule 1) g) v) 3) with:

1) g) v) 3) WQE_i is a <non-join query expression> that immediately contains UNION or UNION ALL. Let $WQEB_i$ be the <query expression body> immediately contained in WQE_i. Let QEL_i and QTR_i be the <query expression> and the <query term> immediately contained in $WQEB_i$. WQN_i shall not be contained in QEL_i, and QTR_i shall be a <query specification>.

5. Rationale: Provide consistent Syntax Rules for comparison operations.

Replace Syntax Rule 11) c) with:

11) c) If the set operator is UNION DISTINCT, EXCEPT DISTINCT, EXCEPT ALL, INTERSECT DISTINCT or INTERSECT ALL, then the declared type of each column of *T1* and *T2* shall not be LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

6. Rationale: Syntax elements don't have descriptor; all tables have column descriptors. Modification of the syntax rules so as to be consistent in the use of definite and indefinite articles.

Replace Syntax Rule 13) with:

13) If a <non-join query term> is a <non-join query primary>, then the declared type of the <non-join query term> is that of the <non-join query primary>. The column descriptor of the *i*-th column of the <non-join query term> is the same as the column descriptor of the *i*-th column of the <non-join query primary>.

7. Rationale: Modification of the syntax rules so as to be consistent in the use of definite and indefinite articles.

Replace Syntax Rule 14) with:

14) If a <non-join query term> immediately contains a set operator, then

a) Case:

i) Let *C* be the <column name> of the *i*-th column of *T1*. If the <column name> of the *i*-th column of *T2* is *C*, then the <column name> of the *i*-th column of *TR* is *C*.

ii) Otherwise, the <column name> of the *i*-th column of *TR* is implementation-dependent and not equivalent to the <column name> of any column, other than itself, of any table referenced by any <table reference> contained in the SQL-statement.

b) The declared type of the *i*-th column of *TR* is determined by applying Subclause 9.3, "Data types of results of aggregations", to the declared types of the *i*-th column of *T1* and the *i*-th column of *T2*. If the *i*-th columns of either *T1* or *T2* are known not nullable, then the *i*-th column of *TR* is known not nullable; otherwise, the *i*-th column of *TR* is possibly nullable.

8. Rationale: Syntax elements don't have descriptor; all tables have column descriptors

Replace Syntax Rule 15) a) with:

15) a) If a <query term> is a <non-join query term>, then the column descriptor of the *i*-th column of the <query term> is the same as the column descriptor of the *i*-th column of the <non-join query term>.

9. Rationale: Modification of the syntax rules so as to be consistent in the use of definite and indefinite articles.

Replace Syntax Rule 15) b) with:

15) b) If a <query term> is a <joined table>, then the column descriptor of the *i*-th column of the <query term> is the same as the column descriptor of the *i*-th column of the <joined table>.

10. Rationale: Correct use of the Case construct in the syntax rule.

Replace Syntax Rule 16) a) with:

16) a) If a <non-join query expression> is a <non-join query term>, then the column descriptor of the *i*-th column of the <non-join query expression> is the same as the column descriptor of the *i*-th column of the <non-join query term>.

11. Rationale: Syntax elements don't have descriptor; all tables have column descriptors

Replace Syntax Rule 17) a) with:

17) a) If a <query expression body> is a <non-join query expression>, then the column descriptors of the *i*-th column of the <query expression body> and of the immediately containing <query expression> are the same as the column descriptor of the *i*-th column of the <non-join query expression>.

12. Rationale: Correct the definition of possibly non-deterministic.

Insert the following Syntax Rule:

18.1) An <explicit table> is *possibly non-deterministic* if the simply contained <table name> identifies a viewed table whose original <query expression> is possibly non-deterministic.

Insert the following Syntax Rule:

19) c.1) UNION, EXCEPT or INTERSECT is specified or implied and there is a column of the result such that the declared types *DT1* and *DT2* of the column in the two operands have corresponding constituents such that one constituent is datetime with time zone and the other is datetime without time zone.

Insert the following Syntax Rule:

19) d) ii) 3) The first or second operand contains a column that has a declared type that is a user-defined type.

13. Rationale: Complete the definition of updatable columns

Replace Syntax Rule 20) d) with:

20) d) If a set operator is specified, then the underlying columns of every *i*-th column of *QE* are the underlying columns of the *i*-th column of *T1* and those of the *i*-th column of *T2*. If a set operator UNION ALL is specified, then a column of that *QE* is called an *updatable column* of *QE* if both its underlying columns of *T1* and *T2* are updatable; otherwise, this column is not updatable. If a set operator UNION DISTINCT, INTERSECT or EXCEPT is specified, then there are no updatable columns.

14. Rationale: Provide consistent Syntax Rules for comparison operations.

Delete Syntax Rules 21) and 22).

15. Rationale: Problems with misuse of syntactic containment.

Replace General Rule 2) a) with:

2) a) Let *n* be the number of <with list element>s WLE_i of the <with list> *WL* immediately contained in *WC*. For *i* ranging from 1 (one) to *n*, and let WQN_i and WQE_i be the <query name>s and the <query expression>s immediately contained in WLE_i. Let WLP_j be the elements of a partitioning of *WL* such that each WLP_j contains all WLE_i that belong to one stratum, and let *m* be the number of partitions. Let the partition dependency graph *PDG* of *WL* be a directed graph such that:

i) Each partition WLP_j of *WL* is represented by exactly one node of *PDG*.

ii) There is an arc from the node representing WLP_j to the node representing WLP_k if and only if WLP_j contains at least one WLE_i, WLP_k contains at least one WLE_h, and WQE_i contains a <query name> referencing WQN_h.

16. Rationale: Provide consistent Conformance Rules for comparison operations.

Replace Conformance Rule 8) with:

8) Without Feature S024, "Enhanced structured types", if any column in the result of a <query expression> is of ST-ordered declared type, then DISTINCT shall not be specified or implied, and neither INTERSECT nor EXCEPT shall be specified.

7.13 <search or cycle clause>

1. Rationale: Problems with misuse of syntactic containment.

Replace Syntax Rule 2) with:

2) Let *WQN* be the <query name>, *WCL* the <with column list>, and *WQE* the <query expression> immediately contained in *WLEC*. Let *WQEB* be the <query expression body> immediately contained in *WQE*. Let *OP* be the set operator immediately contained in *WQEB*. Let *TLO* be the <query expression body> that constitutes the first operand of *OP* and let *TRO* be the <query specification> that (necessarily) constitutes the second operand of *OP*.

a) Let *TROSL* be the <select list> immediately contained in *TRO*. Let *WQNTR* be the <table reference> simply contained in the <from clause> immediately contained in the <table expression> *TROTE* immediately contained in *TRO* such that *WQNTR* immediately contains *WQN*.

Case:

i) If *WQNTR* simply contains a <correlation name>, then let *WQNCRN* be this correlation name.

ii) Otherwise, let *WQNCRN* be *WQN*.

b) Case:

i) If *WLEC* simply contains a <search clause> *SC*, then let *SQC* the <sequence column> and *SO* be the <recursive search order> immediately contained in *SC*. Let *SPL* be the <sort specification list> immediately contained in *SO*.

1) *WCL* shall not contain a <column name> that is equivalent to *SQC*.

2) Every <column name> of *SPL* shall be equivalent to some <column name> contained in *WCL*. No <column name> shall be contained more than once in *SPL*.

3) Case:

A) If *SO* immediately contains DEPTH, then let *SCEX1* be

```
WQNCRN.SQC
```

let *SCEX2* be

```
SQC || ARRAY [ ROW( SPL ) ]
```

and let *SCIN* be

```
ARRAY [ ROW( SPL ) ]
```

B) If *SO* immediately contains BREADTH, then let *SCEX1* be

```
( SELECT OC.*
    FROM ( VALUES ( WQNCRN.SQC ) ) OC( LEVEL, SPL ) )
```

let *SCEX2* be

```
ROW( SQC.LEVEL + 1, SPL )
```

and let *SCIN* be

```
ROW( 0, SPL )
```

ii) If *WLEC* simply contains a <cycle clause> *CC*, then let *CCL* be the <cycle column list>, let *CMC* be the <cycle mark column>, let *CMV* be the <cycle mark value>, let *CMD* be the <non-cycle mark value>, and let *CPA* be the <path column> immediately contained in *CC*.

1) Every <column name> of *CCL* shall be equivalent to some <column name> contained in *WCL*. No <column name> shall be contained more than once in *CCL*.

2) *CMC* and *CPA* shall not be equivalent to each other and not equivalent to any <column name> of *WCL*.

3) The declared type of *CMV* and *CMD* shall be character string of length 1 (one). *CMV* and *CMD* shall be literals and *CMV* shall not be equal to *CMD*.

4) Let *CCEX1* be

```
WQNCRN.CMC, WQNCRN.CPA
```

Let *CCEX2* be

```
CASE
  WHEN ROW( CCL ) IN ( SELECT P.* FROM TABLE( CPA ) P)
    THEN CMV
  ELSE CMD
  END,
CPA || ARRAY [ ROW( CCL ) ]
```

Let *CCIN* be

```
CMD, ARRAY [ ROW( CCL ) ]
```

Let *NCCON1* be

```
CMC <> CMV
```

iii) Case:

1) If *WLEC* simply contains a <search clause> and does not simply contain a <cycle clause>, then let *EWCL* be

WCL, *SQC*

Let *ETLOSL* be

WCL, *SCIN*

Let *ETROSL* be

WCL, *SCEX2*

Let *ETROSL1* be

TROSL, *SCEX1*

Let *NCCON* be

TRUE

2) If *WLEC* simply contains a <cycle clause> and does not simply contain a <search clause>, then let *EWCL* be

WCL, *CMC*, *CPA*

Let *ETLOSL* be

WCL, *CCIN*

Let *ETROSL* be

WCL, *CCEX2*

Let *ETROSL1* be

TROSL, *CCEX1*

Let *NCCON* be

NCCON1

3) If *WLEC* simply contains both a <search clause> and a <cycle clause> *CC*, then:

A) The <column name>s *SQC*, *CMC*, and *CPA* shall not be equivalent to each other.

B) Let *EWCL* be

WCL, *SQC*, *CMC*, *CPA*

Let *ETLOSL* be

WCL, *SCIN*, *CCIN*

Let *ETROSL* be

WCL, *SCEX2*, *CCEX2*

Let *ETROSL1* be

TROSL, *SCEX1*, *CCEX1*

C) Let *NCCON* be

NCCON1

c) *WLEC* is equivalent to the expanded <with list element>

```
WQN( EWCL ) AS
  ( SELECT ETLOSL FROM ( TLO ) TLOCRN( WCL )
    OP
    SELECT ETROSL
      FROM ( SELECT ETROSL1 TROTE ) TROCRN( EWCL )
     WHERE NCCON )
```

7.14 <subquery>

1. Rationale: Not all SQL-statements are atomic.

Insert the following General Rule:

0.1) Let *OLDSEC* be the most recent statement execution context. A new statement execution context *NEWSEC* is established. *NEWSEC* becomes the most recent statement execution context and is atomic.

Replace General Rule 3) with:

3) All savepoints that were established during the existence of *NEWSEC* are destroyed. *NEWSEC* ceases to exist and *OLDSEC* becomes the most recent statement execution context.

8.2 <comparison predicate>

1. Rationale: Provide consistent Syntax Rules for comparison operations.

Replace Syntax Rule 5) a) with:

5) a) If the declared type of X_i or Y_i is LOB-ordered, array-ordered, reference-ordered, or UDT-EC-ordered, then <comp op> shall be either <equals operator> or <not equals operator>.

2. Rationale: Comparison forms and categories do not have to be the same throughout a subtype family. Correct the restriction on comparison functions.

Replace NOTE 126 with:

NOTE 126 — The comparison form and comparison categories included in the user-defined type descriptors of both *UDT1* and *UDT2* are constrained to be the same, and the same as those of all their supertypes. If the comparison category

is either STATE or RELATIVE, then *UDT1* and *UDT2* are constrained to have the same comparison function; if the comparison category is MAP, they are not constrained to have the same comparison function.

3. *Rationale: Syntax Rule 5) b) i) 2) was incorrectly nested.*

Delete Syntax Rule 5) b) i) 2).

Delete Syntax Rule 5) b) i) 3) and the accompanying NOTE.

Add Syntax Rule 5) b) i.1):

5) b) i.1) If the declared types of X_i and Y_i are reference types, then the referenced type of the declared type of X_i and the referenced type of the declared type of Y_i shall have a common supertype.

4. *Rationale: clarify that the choice of an ordering function is not determined by subject routine determination rules. HF1* and *HF2 are <routine name>s and not the SQL-invoked routines themselves.*

Replace General Rule 1) b) iii) 1) with:

1) b) iii) 1) If the comparison category of UDT_x is MAP, then let *HF1* be the <routine name> with explicit <schema name> of the comparison function of UDT_x and let *HF2* be the <routine name> with explicit <schema name> of the comparison function of UDT_y. If *HF1* identifies an SQL-invoked function that is a method, then let *HFX* be *X.HF1*; otherwise, let *HFX* be *HF1* (*X*). If *HF2* identifies an SQL-invoked function that is a method, then let *HFY* be *Y.HF2*; otherwise, let *HFY* be *HF2* (*Y*).

X <comp op> *Y*

has the same result as

HFX <comp op> *HFY*

Replace General Rule 1) b) iii) 2) A) with:

1) b) iii) 2) A) Let *RF* be the <routine name> with explicit <schema name> of the comparison function of UDT_x.

5. *Rationale: Standardise terminology.*

Replace General Rule 3) d) with:

3) d) Depending on the collation, two strings may compare as equal even if they are of different lengths or contain different sequences of characters. When any of the operations MAX, MIN, and DISTINCT reference a grouping column, and the UNION, EXCEPT, and INTERSECT operators refer to character strings, the specific value selected by these operations from a set of such equal values is implementation-dependent.

6. *Rationale: Editorial - typographical error.*

Replace General Rule 4) with:

4) The comparison of two binary string values, X and Y, is determined by comparison of their octets with the same ordinal position. If X_i and Y_i are the values of the i-th octets of X and Y, respectively, and if L_x is the length in octets of X and L_y is the length in octets of Y, then X is equal to Y if and only if $L_x = L_y$ and if $X_i = Y_i$ for all i.

7. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Replace Conformance Rule 1) with:

1) Without Feature T042, "Extended LOB data type support", no field of the declared row type of a <row value expression> that is simply contained in a <comparison predicate> shall be of a LOB-ordered declared type.

Replace Conformance Rule 2) with:

2) Without Feature S024, "Enhanced structured types", no field of the declared type of a <row value expression> that is simply contained in a <comparison predicate> shall be of a declared type that is ST-ordered.

8.3 <between predicate>

1. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Replace Conformance Rule 2) with:

2) Without Feature S024, "Enhanced structured types", no field of the declared type of a <row value expression> that is simply contained in a <between predicate> shall be of a declared type that is ST-ordered.

8.4 <in predicate>

1. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Replace Conformance Rules 2), 3) and 4) with:

2) Without Feature T042, "Extended LOB data type support", no field of the declared row type of a <row value expression> or a <table subquery> contained in an <in predicate> shall be of a LOB-ordered declared type.

3) Without Feature S024, "Enhanced structured types", no field of the declared row type of a <row value expression> or a <table subquery> that is simply contained in an <in predicate> shall be of an ST-ordered declared type.

8.5 <like predicate>

1. Rationale: The non-terminal <character representation> is used inappropriately in some contexts. Standardise terminology.

Replace General Rule 3) b) with:

3) b) Case:

i) If an <escape character> is specified, then:

1) If the length in characters of *ECV* is not equal to 1 (one), then an exception condition is raised: *data exception — invalid escape character*.

2) If there is not a partitioning of the string *PCV* into substrings such that each substring has length 1 (one) or 2, no substring of length 1 (one) is the escape character *ECV*, and each substring of length 2 is the escape character *ECV* followed by either the escape character *ECV*, an <underscore> character, or the <percent> character, then an exception condition is raised: *data exception — invalid escape sequence*. If there is such a partitioning of *PCV*, then in that partitioning, each substring with length 2 represents a single occurrence of the second character of that substring, and is called a *single character specifier*. Each substring with length 1 (one) that is the <underscore> character represents an *arbitrary character specifier*. Each substring with length 1 (one) that is the <percent> character represents an *arbitrary string specifier*. Each substring with length 1 (one) that is neither the <underscore> character nor the <percent> character represents the character that it contains, and is called a *single character specifier*.

ii) If an <escape character> is not specified, then each <underscore> character in *PCV* represents an arbitrary character specifier, each <percent> character in *PCV* represents an arbitrary string specifier, and each character in *PCV* that is neither the <underscore> character nor the <percent> character represents itself, and is called a *single character specifier*.

Delete General Rule 3) c).

Replace General Rule 3) d) ii) 1), General Rule 3) d) ii) 2), General Rule 3) d) ii) 3), and General Rule 3) d) ii) 4) with:

3) d) ii) 1) A substring of *MCV* is a sequence of 0 (zero) or more contiguous characters of *MCV* and each character of *MCV* is part of exactly one substring.

3) d) ii) 2) If the *i*-th substring of *PCV* is an arbitrary character specifier, the *i*-th substring of *MCV* is any single character.

3) d) ii) 3) If the *i*-th substring of *PCV* is an arbitrary string specifier, then the *i*-th substring of *MCV* is any sequence of 0 (zero) or more characters.

3) d) ii) 4) If the *i*-th substring of *PCV* is a single character specifier, then the *i*-th substring of *MCV* contains exactly 1 (one) character that is equal to the character represented by the single character specifier, according to the collation of the <like predicate>.

2. *Rationale: Clarify the distinction between character sets and character repertoires.*

Replace General Rules 4) b) and c) with:

4) b) <percent> in the context of an <octet like predicate> has the same bit pattern as the encoding of a <percent> in the SQL_TEXT character set.

c) <underscore> in the context of an <octet like predicate> has the same bit pattern as the encoding of an <underscore> in the SQL_TEXT character set.

8.6 <similar predicate>

1. *Rationale: Correct misplaced BNF rules in <regular expression>.*

Replace the Format for <regular character set> with:

```
<regular character set> ::=
  <underscore>
| <left bracket> <character enumeration>... <right bracket>
| <left bracket> <circumflex> <character enumeration>... <right bracket>
```

Replace the Format for <character enumeration> with:

```
<character enumeration> ::=
  <character specifier>
| <character specifier> <minus sign> <character specifier>
| <left bracket> <colon>
    <regular character set identifier>
  <colon> <right bracket>
```

2. *Rationale: Standardise terminology.*

Replace Syntax Rule 4) with:

4) Case:

a) If <escape character> is not specified, then the collation used for the <similar predicate> is determined by Table 3, ''Collating sequence usage for comparisons'', taking <character match value> as comparand 1 (one) and <similar pattern> as comparand 2.

b) Otherwise, let C1 be the coercibility characteristic and collation of the <character match value>, and C2 be the coercibility characteristic and collation of the <similar pattern>. Let C3 be the resulting coercibility characteristic and collation as determined by Table 2, ''Collating coercibility rules for dyadic operators'', taking C1 as the operand 1 (one) coercibility and C2 as the operand 2 coercibility. The collation used for the <similar predicate> is determined by Table 3, ''Collating sequence usage for comparisons'', taking C3 as the coercibility characteristic and collation of comparand 1 (one) and <escape character> as comparand 2.

It is implementation-defined, whether all, some, or no collations other than the default collation for the character set of the <character match value> can be used as the collation of the <similar predicate>.

3. *Rationale: Correct misplaced BNF rules in <regular expression>.*

Delete Syntax Rule 7).

4. *Rational: "P" is referenced in General Rules 5) a), 6) e), 6) g), and 6) i) but is not defined. Standardise terminology.*

Replace General Rule 5) a) with:

5) a) If the enumeration is specified in the form "<character specifier> <minus sign> <character specifier>", then the set of all characters that collate greater than or equal to the character represented by the left <character specifier> and less than or equal to the character represented by the right <character specifier>, according to the collation of the pattern *PCV*.

Replace General Rule 6) e) with:

6) e) L (<percent>)

is the set of all strings of any length (zero or more) from the character set of the pattern *PCV*.

Replace General Rule 6) g) with:

6) g) L (<underscore>)

is the set of all strings of length 1 (one) from the character set of the pattern *PCV*.

Replace General Rule 6) i) with:

6) i) L (<left bracket> <circumflex> <character enumeration> <right bracket>)

is the set of all strings of length 1 (one) with characters from the character set of the pattern *PCV* that are not contained in the set of characters in the <character enumeration>.

5. *Rationale: Standardise terminology.*

Replace General Rule 7) with:

7) The <similar predicate>

CM SIMILAR TO *SP*

is *True*, if there exists at least one element *X* of L(*R*) that is equal to *MCV* according to the collating sequence of the <similar predicate>; otherwise, it is *False*.

NOTE 134 – The <similar predicate> is defined differently from equivalent forms of the LIKE predicate. In particular, blanks at the end of a pattern and collating sequences are handled differently.

6. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Insert the following Conformance Rule:

2) Without Feature T042, "Extended LOB data type support", a <character value expression> contained in a <similar predicate> shall not be of declared type CHARACTER LARGE OBJECT.

8.7 <null predicate>

1. Rationale: Specify the result for the case when the value of the <row value expression> is the null value.

Replace General Rule 2) with:

2) Case:

a) If *R* is the null value, then "*R* IS NULL" is *True*.

b) Otherwise:

i) The value of "*R* IS NULL" is

Case:

a) If the value of every field in *R* is the null value, then *True*.

b) Otherwise, *False*.

ii) The value of "*R* IS NOT NULL" is

Case:

a) If the value of no field in *R* is the null value, then *True*.

b) Otherwise, *False*.

Delete General Rule 3).

8.8 <quantified comparison predicate>

1. Rationale: Provide consistent Conformance Rules for comparison operations.

Replace the Conformance Rules 1) and 2) with:

1) Without Feature T042, "Extended LOB data type support", no field of the declared row type of a <row value expression> or a <table subquery> contained in a <quantified comparison predicate> shall be of a LOB-ordered declared type.

2) Without Feature S024, "Enhanced structured types", no field of the declared row type of a <row value expression> shall be of an ST-ordered declared type.

8.10 <unique predicate>

1. Rationale: Clarify that unordered types do not support <unique predicate>.

In Syntax Rules, replace "None" with:

1) Each column of user-defined type in the result of the <table subquery> shall have a comparison type.

2. Rationale: Provide consistent Syntax Rules for comparison operations.

Insert the following Syntax Rule:

2) No column of the result of the <table subquery> shall be of a declared type that is LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

3. Rationale: Use the notion of distinct rather than equality

Replace General Rule 2) with:

2) If there are no two rows in *T* such that the value of each column in one row is non-null and is not distinct from the value of the corresponding column in the other row, then the result of the <unique predicate> is *True*; otherwise, the result of the <unique predicate> is *False*.

4. Rationale: Provide consistent Conformance Rules for comparison operations.

Replace Conformance Rule 2) with:

2) Without Feature S024, "Enhanced structured types", no column of the result of the <table subquery> shall be of ST-ordered declared type.

8.11 <match predicate>

1. Rationale: Editorial - typographical error.

Replace Syntax Rule 1) with:

1) The row type of the <row value expression> and the row type of the <table subquery> shall be comparable.

2. Rationale: Specify the result for the case when the value of the <row value expression> is the null value.

Replace General Rule 2) with:

2) If SIMPLE is specified or implicit, then

Case:

a) If *R* is the null value, then the <match predicate> is *True*.

b) Otherwise:

i) If the value of some field value in *R* is the null value, then the <match predicate> is *True*.

ii) If the value of no field in *R* is the null value, then

Case:

1) If UNIQUE is not specified and there exists a row RT_i of the <table subquery> such that

$R = RT_i$

then the <match predicate> is *True*.

2) If UNIQUE is specified and there is exactly one row RT_i in the result of evaluating the <table subquery> such that

$R = RT_i$

then the <match predicate> is *True*.

3) Otherwise, the <match predicate> is *False*.

Replace General Rule 3) with:

3) If PARTIAL is specified, then

Case:

a) If *R* is the null value, then the <match predicate> is *True*.

b) Otherwise:

i) If the value of every field in *R* is the null value, then the <match predicate> is *True*.

ii) Otherwise,

Case:

1) If UNIQUE is not specified and there exists a row RT_i of the <table subquery> such that each non-null value of *R* equals its corresponding value in RT_i, then the <match predicate> is *True*.

2) If UNIQUE is specified and there is exactly one row RT_i in the result of evaluating the <table subquery> such that each non-null value of *R* equals its corresponding value in RT_i, then the <match predicate> is *True*.

3) Otherwise, the <match predicate> is *False*.

Replace General Rule 4) with:

4) If FULL is specified, then

Case:

a) If *R* is the null value, then the <match predicate> is *True*.

b) Otherwise:

i) If the value of every field in *R* is the null value, then the <match predicate> is *True*.

ii) If the value of no field in *R* is the null value, then

Case:

1) If UNIQUE is not specified and there exists a row RT_i of the <table subquery> such that

$R = RT_i$

then the <match predicate> is *True*.

2) If UNIQUE is specified and there exists exactly one row RT_i in the result of evaluating the <table subquery> such that

$R = RT_i$

then the <match predicate> is *True*.

3) Otherwise, the <match predicate> is *False*.

iii) Otherwise, the <match predicate> is *False*.

3. Rationale: Provide consistent Conformance Rules for comparison operations.

Replace Conformance Rule 2) with:

2) Without Feature S024, "Enhanced structured types", no field of the declared row type of the <row value expression> shall be of an ST-ordered declared type and no column of the result of the <table subquery> shall be of an ST-ordered declared type.

3) Without Feature T042, "Extended LOB data type support", no field of the declared row type of the <row value expression> shall be of a LOB-ordered declared type.

8.12 <overlaps predicate>

1. Rationale: Specify the result for the case when the value of either of the two <row value expression>s is the null value.

Insert the following General Rule:

0.1) If the value of <row value expression 1> is the null value or if the value of <row value expression 2> is the null value, then the result of <overlaps predicate> is *Unknown* and no further General Rules of this Subclause are applied.

2. *Rationale: Create a separate Feature for <overlaps predicate>.*

Replace Conformance Rule 1) with:

1) Without Feature F053, "OVERLAPS predicate", conforming SQL language shall not contain any <overlaps predicate>.

8.13 <distinct predicate>

1. *Rationale: Use the notion of distinct.*

Replace General Rules 1), 2) and 3) with:

1) The result of <distinct predicate> is *True* if the value of <row value expression 3> is distinct from the value of <row value expression 4>; otherwise the result is *False*.
NOTE 136.1 — "distinct" is defined in Subclause 3.1.5, "Definitions provided in Part 2".

2. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Replace Conformance Rule 2) with:

2) Without Feature S024, "Enhanced structured types", no field of the declared row type of either <row value expression> shall be of an ST-ordered declared type.

3) Without Feature T042, "Extended LOB data type support", no field of the declared row type of either <row value expression> shall be of a LOB-ordered declared type.

8.14 <type predicate>

1. *Rationale: Clarify that a type that is either inclusively specified or exclusively specified has to be a subtype of the declared type of the <user-defined type value expression>.*

Replace Syntax Rule 3) with:

3) If *T* is a type specified by <inclusive user-defined type specification> or <exclusive user-defined type specification>, then *T* shall be a subtype of the declared type of the <user-defined type value expression>.

9.0 Determination of identical values

1. *Rationale: Specify the meaning of "identical"*

Insert the following subclause:

9.0 Determination of identical values

Function

Determine whether two instances of values are identical, that is to say are occurrences of the same value.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *V1* and *V2* be two values specified in an application of this subclause.

NOTE 137.1 — This subclause is invoked implicitly wherever the word *identical* is used of two values.

2) Case:

a) If *V1* and *V2* are both null, then *V1* is identical to *V2*.

b) If *V1* is null and *V2* is not null or *V1* is not null and *V2* is null, then *V1* is not identical to *V2*.

c) If *V1* and *V2* are of comparable predefined types, then

Case:

i) If *V1* and *V2* are character strings then

Case:

1) If *V1* is not distinct from *V2* and CAST (*V1* AS BIT (BIT_LENGTH (*V1*)) is not distinct from CAST (*V2* AS BIT (BIT_LENGTH (*V2*)), then *V1* is identical to *V2*.

2) Otherwise, *V1* is not identical to *V2*.

ii) If *V1* and *V2* are time with time zone or timestamp with time zone and are not distinct and their time zone fields are not distinct, then *V1* is identical to *V2*.

iii) Otherwise, *V1* is identical to *V2* if and only if *V1* is not distinct from *V2*.

d) If *V1* and *V2* are of constructed types, then

Case:

i) If *V1* and *V2* are rows and their respective fields are identical, then *V1* is identical to *V2*.

ii) If *V1* and *V2* are arrays and have the same cardinality and elements in the same ordinal position in the two arrays are identical, then *V1* is identical to *V2*.

iii) If *V1* and *V2* are references and *V1* is not distinct from *V2*, then *V1* is identical to *V2*.

iv) Otherwise, *V1* is not identical to *V2*.

e) If *V1* and *V2* are of the same most specific type, *MST*, and *MST* is a user-defined type, then:

Case:

i) If *MST* is a distinct type, whose source type is *SDT* and the results of *SDT* (*V1*) and *SDT* (*V2*) are identical, then *V1* is identical to *V2*.

ii) If *MST* is a structured type and, for every observer function *O* defined for *MST*, the results of the invocations *O* (*V1*) and *O* (*V2*) are identical, then *V1* is identical to *V2*.

iii) Otherwise, *V1* is not identical to *V2*.

f) Otherwise, *V1* is not identical to *V2*.

Conformance Rules

None.

9.1 Retrieval assignment

1. *Rationale: Clarify assignable and comparable.*

Replace Syntax Rules 1) and 2) with:

1) Let *T* and *V* be a TARGET and VALUE specified in an application of this Subclause. Let the declared types of *T* and *V* be *TD* and *SD* respectively.

2) If *TD* is bit string, binary string, numeric, boolean, datetime, interval, or a user-defined type, then either *SD* shall be assignable to *TD* or there shall exist a user-defined cast function *UDCF* from *SD* to *TD*.

2. *Rationale: Restore the facility of assignments involving character strings if there is a user-defined cast function. Clarify assignable and comparable*

Replace Syntax Rule 3) with:

3) If the declared type of *T* is character string, then

Case:

a) If *T* is either a locator parameter of an external routine, a locator variable, or a host parameter that is a character large object locator parameter, then *SD* shall be assignable to *TD*.

b) Otherwise, either the *SD* shall be assignable to *TD* or there shall exist a user-defined cast function *UDCF* from *SD* to *TD*.

9.2 Store assignment

1. Rationale: Clarify assignable and comparable.

1) Let *T* and *V* be a TARGET and VALUE specified in an application of this Subclause. Let the declared types of *T* and *V* be *TD* and *SD* respectively.

2) If the declared type of *T* is character string, bit string, binary string, numeric, boolean, datetime, interval, or a user-defined type, then either *SD* shall be assignable to *TD* or there shall exist a user-defined cast function *UDCF* from *SD* to *TD*.

9.3 Data types of results of aggregations

1. Rationale: Clarify the distinction between character sets and character repertoires.

Replace Syntax Rule 3) a) with:

3) a) If any of the data types in *DTS* is character string, then all data types in *DTS* shall be character string, and all of them shall have the same character repertoire. The character set of the result is the character set of the data type in DTS that has the character encoding form with the highest precedence. The collating sequence and the coercibility characteristic are determined as specified in Table 2, ''Collating coercibility rules for dyadic operators''.

Case:

i) If any of the data types in *DTS* is character large object string, then the result data type is character large object string with maximum length in characters equal to the maximum of the lengths in characters and maximum lengths in characters of the data types in *DTS*.

ii) If any of the data types in *DTS* is variable-length character string, then the result data type is variable-length character string with maximum length in characters equal to the maximum of the lengths in characters and maximum lengths in characters of the data types in *DTS*.

iii) Otherwise, the result data type is fixed-length character string with length in characters equal to the maximum of the lengths in characters of the data types in *DTS*.

2. Rationale: Editorial.

Replace Syntax Rule 3) i) with:

i) If any data type in *DTS* is a row type, then each data type in *DTS* shall be a row type with the same degree and the data type of each field in the same ordinal position of every row type shall be comparable. The result data type is a row defined by an ordered sequence of (<field name>, data type) pairs FD_i, where data type is the data type resulting from the application of this Subclause to the set of data types of fields in the same ordinal position as FD_i in every row type in *DTS* and <field name> is determined as follows:

Case:

i) If the names of fields in the same ordinal position as FD_i in every row type in *DTS* is *F*, then the <field name> in FD_i is *F*.

ii) Otherwise, the <field name> in FD_i is implementation-dependent.

3. *Rationale: Correct reference to undefined <scope table name list>.*

Replace Syntax Rule 3) k) i) with:

3) k) i) If the data type descriptor of every data type in *DTS* includes the name of a referenceable table identifying the scope of the reference type, and every such name is equivalent to some name *STN*, then result data type is:

RT SCOPE (*STN*)

9.5 Type precedence list

1. *Rationale: Editorial.*

Replace Syntax Rule 2) with:

2) The *type precedence list TPL* of *DT* is a list of type designators as specified in the Syntax Rules of this Subclause.

2. *Rationale: Correct the type precedence list for reference types.*

Replace Syntax Rule 18) with:

18) If *DT* is a reference type, then let *n* be the number of elements in the type precedence list for the referenced type of *DT*. For *i* ranging from 1 (one) to *n*, let KAW_i be the *i*-th such element. *TPL* is

REF(KAW_1),REF(KAW_2), ..., REF(KAW_n)

10.3 <path specification>

1. *Rationale: Clarify the semantics of built-in functions.*

Delete Syntax Rule 2).

10.4 <routine invocation>

1. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 4) with:

4) Case:

a) If *RI* is immediately contained in a <call statement>, then an SQL-invoked routine *R* is a *possibly candidate routine* for *RI* (henceforth, simply "possibly candidate routine") if *R* is an SQL-invoked

procedure and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

b) If *RI* is immediately contained in a <method selection>, then an SQL-invoked routine *R* is a *possibly candidate routine* for *RI* if *R* is an instance SQL-invoked method and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

c) If *RI* is immediately contained in a <constructor method selection>, then an SQL-invoked routine *R* is a *possibly candidate routine* for *RI* if *R* is an SQL-invoked constructor method and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

d) If *RI* is immediately contained in a <static method selection>, then an SQL-invoked routine *R* is a possibly candidate routine for *RI* if *R* is a static SQL-invoked method and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN* and the method specification descriptor *MSD* is included in a user-defined type descriptor for *UDTSM* or for some supertype of *UDTSM*.

e) Otherwise, an SQL-invoked routine *R* is a *possibly candidate routine* for *RI* if *R* is an SQL-invoked function that is not an SQL-invoked method and the <qualified identifier> of the <routine name> of *R* is equivalent to the <qualified identifier> of *RN*.

2. *Rationale: Change "user-defined data type" to "user-defined type".*

Replace Syntax Rule 6) b) iii) 2) A) with:

6) b) iii) 2) A) If the declared type of P_i is a user-defined type, then:
I) Let ST_i be the set of subtypes of the declared type of A_i.
II) The type designator of the declared type of P_i shall be in the type precedence list of the data type of some type in ST_i.
NOTE 149 — "type precedence list" is defined in Subclause 9.5, "Type precedence list determination".

3. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 7) b) i) 1) A) with:

7) b) i) 1) A) If *RI* is immediately contained in a <method selection>, <static method selection> or a <constructor method selection>, then let *DP* be *TP*.

4. *Rationale: Editorial.*

Replace Syntax Rule 7) b) i) 2) C) II) with:

7) b) i) 2) C) II) If the routine descriptor of *R1* includes a STATIC indication, then there is no other invocable routine *R2* for which the user-defined type described by the user-defined type descriptor that includes the routine descriptor of *R2* is a subtype of the user-defined type described by the user-defined type descriptor that includes the routine descriptor of *R1*.

5. *Rationale: Clarify the semantics of built-in functions.*

Replace Syntax Rule 7) b) ii) with:

7) b) ii) If *RN* contains a <schema name> *SN*, then *SN* shall be the <schema name> of a schema *S*. The *subject routine* of *RI* is the invocable routine (if any) contained in *S*.

6. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 8) b) i) 1) A) with:

8) b) i) 1) A) If *RI* is immediately contained in a <method selection>, <static method selection> or a <constructor method selection>, then let *DP* be *TP*.

7. *Rationale: Clarify the semantics of built-in functions.*

Replace Syntax Rule 8) b) ii) with:

8) b) ii) If *RN* contains a <schema name> *SN*, then *SN* shall be the <schema name> of a schema *S*. The candidate routines of *RI* are the invocable routines (if any) contained in *S*.

8. *Rationale: The current handling of output parameter in routine invocation is incomplete. It does not cover all alternatives of <target specification>. Prohibit updating new transition variables in AFTER triggers*

Replace Syntax Rule 8) c) i) 4) with:

8) c) i) 4) For each P_i that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, A_i shall be a <target specification>.

A.0) If *RI* is contained in a <triggered SQL statement> of an AFTER trigger, then A_i shall not be a <column reference>.

A) If A_i is a <host parameter specification>, then P_i shall be assignable to A_i, according to the Syntax Rules of Subclause 9.1, "Retrieval assignment", with A_i and P_i as *TARGET* and *VALUE*, respectively.

B) If A_i is an <SQL parameter reference> or a <column reference>, then P_i shall be assignable to A_i, according to the Syntax Rules of Subclause 9.2, "Store assignment", with A_i and P_i as *TARGET* and *VALUE*, respectively.
NOTE 150.1 — The <column reference> can only be a new transition column reference.

9. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 9) with:

9) If *SR* is a constructor function, then *RI* shall be simply contained in a <new invocation>.

10. Rationale: Cater properly for null subject argument.

Insert the following General Rule:

2) b) ii.1) If *SR* is type preserving and the null value is substituted for the result parameter, then

Case:

1) If *SR* is a mutator function, then an exception condition is raised: *data exception — null value substituted for mutator subject parameter.*

2) Otherwise, the value of *RI* is the null value and the remaining General Rules of this subclause do not apply.

11. Rationale: Cater properly for null subject argument.

Insert the following General Rule:

2) b) iii) 1) A.0) If V_1 is the null value, then the value of *RI* is the null value and the remaining General Rules of this Subclause are not applied.

12. Rationale: Change "user-defined data type" to "user-defined type" and the determination of overriding methods should be based on data type identity rather than on data type compatibility.

Replace General Rule 2) b) iii) 1) A) IV) with:

2) b) iii) 1) A) IV) For *j* varying from 2 to *N*, the Syntax Rules of Subclause 10.17, ''Data type identity'', are applied with the declared type of PM_j and the declared type of PSR_j.
NOTE 152 — *SR* is an element of the set *SM*.

13. Rationale: A value does not have a declared type.

Replace General Rule 3) b) with:

3) b) Otherwise, let CPV_i be an implementation-defined value of most specific type T_i.

14. Rationale: Clarify the semantics of built-in functions.

Delete General Rule 4).

15. Rationale: Address requirement for multiple diagnostics areas

Insert the following General Rule:

6) b.1) The diagnostics area stack in *CSC* is copied to *RSC* and the General Rules of Subclause 19.2, "Pushing and popping diagnostics areas" are applied with PUSH as *OPERATION* and the diagnostics area stack in *RSC* as *STACK*.

16. Rationale: Clarify the semantics of SQL-data access indication.

Replace General Rule 6) e) with:

6) e) Case:

i) If the SQL-data access indication of *CSC* specifies possibly contains SQL and *R* possibly reads SQL-data, or *R* possibly modifies SQL-data, then

1) If *R* is an external routine, then an exception condition is raised: *external routine exception — reading SQL-data not permitted*.

2) Otherwise, an exception condition is raised: *SQL routine exception — reading SQL-data not permitted*.

ii) If the SQL-data access indication of *CSC* specifies possibly reads SQL-data and *R* possibly modifies SQL-data, then:

1) If *R* is an external routine, then an exception condition is raised: *external routine exception — modifying SQL-data not permitted*.

2) Otherwise, an exception condition is raised: *SQL routine exception — modifying SQL-data not permitted*.

Insert the following General Rule:

6) e.1) Case:

i) If *R* does not possibly contain SQL, then set the SQL-data access indication in the routine execution context of *RSC* to does not possibly contain SQL.

ii) If *R* possibly contains SQL, then set the SQL-data access indication in the routine execution context of *RSC* to possibly contains SQL.

iii) If *R* possibly reads SQL-data, then set the SQL-data access indication in the routine execution context of *RSC* to possibly reads SQL-data.

iv) If *R* possibly modifies SQL-data, then set the SQL-data access indication in the routine execution context of *RSC* to possibly modifies SQL-data.

Delete General Rule 7).

17. Rationale: Cater properly for null subject argument.

Delete General Rule 8) a).

18. Rationale: Clarify the semantics of SQL-data access indication.

Replace General Rule 8) c) iv) with:

8) c) iv) If the SQL-data access indication of *RSC* specifies possibly contains SQL, and, before the completion of the execution of the <SQL routine body> of *R*, an attempt is made to execute an

SQL-statement that possibly reads SQL-data, or an attempt is made to execute an SQL-statement that possibly modifies SQL-data, then an exception condition is raised: *SQL routine exception — reading SQL-data not permitted.*

19. *Rationale: Clarify the semantics of SQL-data access indication.*

Replace General Rule 8) c) v) with:

8) c) v) If the SQL-data access indication of *RSC* specifies possibly reads SQL-data, and, before the completion of the execution of the <SQL routine body> of *R*, an attempt is made to execute an SQL statement that possibly modifies SQL-data, then an exception condition is raised: *SQL routine exception — modifying SQL-data not permitted.*

20. *Rationale: Remove erroneous specification concerning re-raising of completion condition.*

Delete General Rule 8) c) vi)

21. *Rationale: Fix the subscripts in the references to CPV in General Rule 9) b) i) 1) C) and General Rule 9) b) i) 2) B).*

Replace General Rule 9) b) i) 1) C) with:

9) b) i) 1) C) For *i* ranging from (*PN*+1)+1 to (*PN*+1)+*N*, the *i*-th entry in *ESPL* is the SQL indicator argument corresponding to $CPV_{i-(PN+1)}$.

22. *Rationale: Fix the subscripts in the references to CPV in General Rule 9) b) i) 1) C) and General Rule 9) b) i) 2) B).*

Replace General Rule 9) b) i) 2) B) with:

9) b) i) 2) B) For *i* ranging from *PN*+1 to *PN*+*N*, the *i*-th entry in *ESPL* is the SQL indicator argument corresponding to CPV_{i-PN}.

23. *Rationale: Cater properly to null-call functions with PARAMETER STYLE GENERAL.*

Replace General Rule 9) c) with:

9) c) If *R* specifies PARAMETER STYLE GENERAL, then the effective SQL parameter list *ESPL* of *R* is set as follows:

i) If *R* is not a null-call function and, for *i* ranging from 1 (one) to *PN*, CPV_i is the null value then an exception condition is raised: *external routine invocation exception — null value not allowed.*

ii) For *i* ranging from 1 (one) to *PN*, if no CPV_i is the null value, then for *j* ranging from 1 (one) to *PN*, the *j*-th entry in *ESPL* is set to CPV_i

24. *Rationale: Cater properly for null subject argument.*

Delete General Rule 9) f) i).

25. Rationale: Clarify the semantics of SQL-data access indication.

Insert the following General Rule:

9) f) iii) 2.1) If the SQL-data access indication of *RSC* specifies does not possibly contain SQL, and before the completion of any execution of *P*, an attempt is made to execute an SQL-statement, then an exception condition is raised: *external routine exception — containing SQL not permitted.*

26. Rationale: Clarify the semantics of SQL-data access indication.

Delete General Rule 9) f) iii) 4).

27. Rationale: Clarify the semantics of SQL-data access indication.

Replace General Rule 9) f) iii) 5) with:

9) f) iii) 5) If the SQL-data access indication of *RSC* specifies possibly contains SQL, and, before the completion of any execution of *P*, an attempt is made to execute an SQL-statement that possibly reads SQL-data, or an attempt is made to execute an SQL-statement that possibly modifies SQL-data, then an exception condition is raised: *external routine exception — reading SQL-data not permitted.*

28. Rationale: Clarify the semantics of SQL-data access indication.

Replace General Rule 9) f) iii) 6) with:

9) f) iii) 6) If the SQL-data access indication of *RSC* specifies possibly reads SQL-data, and, before the completion of any execution of *P*, an attempt is made to execute an SQL-statement that possibly modifies SQL-data, then an exception condition is raised: *external routine exception — modifying SQL-data not permitted.*

29. Rationale: Correct rules referencing "exception data item" to be only applicable to external routines that specify PARAMETER STYLE SQL. Remove restrictions on SQLSTATE values generated by external routines

Replace General Rule 9) g) ii) with:

9) g) ii) For *i* varying from 1 (one) to *EN*, the value of ESP_i is set to the value of PD_i. If *R* specifies PARAMETER STYLE SQL, then:

Case:

1) If the exception data item has the value '00000', then the execution of *P* was successful.

2) If the first two characters of the exception data item are equal to the SQLSTATE condition code class value for *warning*, then a completion condition is raised: *warning*, using a subclass code equal to the final three characters of the value of the exception data item.

3) Otherwise, an exception condition is raised using a class code equal to the first two characters of the value of the exception data item and a subclass code equal to the final three character of the value of the exception data item.

30. Rationale: Address requirement for multiple diagnostics areas

Replace General Rule 9) g) iii) with:

9) g) iii) If the exception data item is not '00000' and *R* specified PARAMETER STYLE SQL, then the message text item is stored in the first diagnostics area.

31. Rationale: Allow the effective result data item to be set for null-call functions when the return value is not null.

Replace General Rule 9) h) i) 4) A) with:

9) h) i) 4) A) Case:

I) If *R* is not an array-returning external function, *R* specifies PARAMETER STYLE SQL, and entry (*PN* + 1) + *N* + 1 in *ESPL* (that is, SQL indicator argument *N* + 1 corresponding to the result data item) is not negative, then let *ERDI* be the value of the result data item.

II) If *R* is an array-returning external function, and *R* specifies PARAMETER STYLE SQL, then let *ERDI* be *AR*.

III) If *R* specifies PARAMETER STYLE GENERAL, then let *ERDI* be the value returned from *P*.

NOTE 156 — The value returned from *P* is passed to the SQL-implementation in an implementation-dependent manner. An argument value list entry is not used for this purpose.

32. Rationale: Type-preserving functions must return values that have the same type as the most specific type of the argument substituted for the RESULT parameter. For handling the return value add the case when neither the <result cast> has a locator indication nor the data type of the result value is a user-defined type.

Replace General Rule 9) h) i) 4) B) ii) 2) with:

9) h) i) 4) B) II) 2) Otherwise,

Case:

a) If *CRT* is a user-defined type, then:

i) Let *TSF* be the SQL-invoked routine identified by the specific name of the to-sql function associated with the result of *R*.

ii) Case:

1) If *TSF* is an SQL-invoked method, then:

A) If *R* is a type-preserving function, then let *MAT* be the most specific type of the value of the argument substituted for the result SQL parameter of *R*; otherwise, let *MAT* be *CRT*.

B) The General Rules of this Subclause are applied with a static SQL argument list whose first element is the value returned by the invocation of:

MAT()

and whose second element is *ERDI*, and the subject routine *TSF*.

2) Otherwise, the General Rules of this Subclause are applied with a static SQL argument list that has a single SQL argument that is *ERDI*, and the subject routine *TSF*.

iii) Let *RDI* be the result of invocation of *TSF*.

b) Otherwise, let *RDI* be *ERDI*.

33. *Rationale: Check for the most specific type of the returned value from type-preserving functions. The current handling of output parameter in routine invocation is incomplete. It does not cover all alternatives of <target specification>.*

Replace General Rule 10) with:

10) Case:

a) If *R* is an SQL-invoked function, then:

i) If *R* is type preserving, then:

1) Let *MAT* be the most specific type of the value of the argument substituted for the result SQL parameter of *R*.

2) *If RV* is not the null value and if the most specific type of *RV* is not compatible with *MAT*, then an exception condition is raised: *data exception — most specific type mismatch.*

ii) Let *ERDT* be the effective returns data type of the <routine invocation>.

iii) Let the result of the <routine invocation> be the result of assigning *RV* to a target of declared type *ERDT* according to the rules of Subclause 9.2, "Store assignment".

b) Otherwise, for each SQL parameter P_i of *R* that is an output SQL parameter or both an input SQL parameter and an output SQL parameter, let TS_i be the <target specification> of the corresponding <SQL argument> A_i.

Case:

i) If TS_i is a <host parameter specification>, then CPV_i is assigned to TS_i according to the rules of Subclause 9.1, "Retrieval assignment".

ii) If TS_i is an <SQL parameter reference>, or a <column reference>, then CPV_i is assigned to TS_i according to the rules of Subclause 9.2, "Store assignment".
NOTE 157.1 — The <column reference> can only be a new transition column reference.

34. *Rationale: Ensure that the complete result set is available for scrollable cursors*

Replace General Rule11) e) with:

11) e) Case:

i) If $FRCN_i$ is a scrollable cursor, then let NXT_i, $1 \le i \le RTN$, be 1 (one).

ii) Otherwise, let NXT_i,$1 \le i \le RTN$, be the ordinal number of the row of RS_i that would be retrieved if the following SQL-statement were executed:

```
FETCH NEXT FROM FRCN_i INTO . . .
```

35. *Rationale: Allow information about warnings generated during evaluation of routine invocations to be visible via GET DIAGNOSTICS.*

Replace General Rule 12) d) with:

12) d) Set the value of the current transaction condition area limit in *CSC* to the value of the current transaction condition area limit *CAL* in *RSC*.

Insert the following General Rule:

12) d.1) For each occupied condition area *CA* in the first diagnostics area of *RSC*, if the value of RETURNED_SQLSTATE in *CA* does not represent *successful completion*, then

Case:

i) If the number of occupied condition areas in the first diagnostics area *DA1* in *CSC* is less than *CAL*, then *CA* is copied to the first vacant condition area in *DA1*.
NOTE 157.2 — This causes the first vacant condition area in *DA1* to become occupied.

ii) Otherwise, the value of MORE in the statement information area of *DA1* is set to 'Y'.

10.5 <privileges>

1. *Rationale: Clarify that ALL PRIVILEGES, UNDER and USAGE can be specified for <object name>s that are <schema-resolved user-defined type name>s.*

Replace Syntax Rule 3) with:

3) If <object name> specifies a <domain name>, <collation name>, <character set name>, <transliteration name>, or <schema-resolved user-defined type name>, then <privileges> may specify USAGE. Otherwise, USAGE shall not be specified.

2. *Rationale: Clarify that ALL PRIVILEGES and EXECUTE can be specified for <object name>s that are SQL-invoked routines.*

Replace Syntax Rule 7) with:

7) If the object identified by <object name> of the <grant statement> or <revoke statement> is an SQL-invoked routine, then <privileges> may specify EXECUTE; otherwise, EXECUTE shall not be specified.

3. *Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".*

Replace Conformance Rule 9) with:

9) Without Feature F461, "Named character sets", in conforming SQL language, an <object name> shall not specify CHARACTER SET.

10.6 <character set specification>

1. *Rationale: Align Syntax Rules with the Concepts.*

Replace Syntax Rule 3) with:

3) The <standard character set name>s shall include: SQL_CHARACTER and all those specified in Subclause 4.2.4, "Named character sets" as defined by other standards.

2. *Rationale: Standardise terminology.*

Replace General Rules 2), 3) and 4)as shown here:

2) A <standard character set name> specifies the name of a character set that is defined by a national or international standard. The character repertoire of CS is defined by the standard defining the character set identified by that <standard character set name>. The default collation of the character set is defined by the order of the characters in the standard and has the PAD SPACE characteristic.

3) An <implementation-defined character set name> specifies the name of a character set that is implementation-defined. The character repertoire of CS is implementation-defined. The default collation of the character set and whether the collation has the NO PAD characteristic or the PAD SPACE characteristic is implementation-defined.

4) A <user-defined character set name> identifies a character set whose descriptor is included in some schema whose <schema name> is not equivalent to INFORMATION_SCHEMA.
NOTE 215 — The default collation of the character set is defined as in Subclause 11.31, "<character set definition>".

3. *Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".*

Replace Conformance Rule 1) with:

1) Without Feature F461, "Named character sets", conforming SQL language shall not contain a <character set specification>.

10.7 <specific routine designator>

1. *Rationale: Use the correct BNF (<user-defined type name> instead of <user-defined type>).*

In the Format, replace the production for <specific routine designator> with:

```
<specific routine designator> ::=
        SPECIFIC <routine type> <specific name>
      | <routine type> <member name> [ FOR <user-defined type name> ]
```

2. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

In the Format, replace the production for <routine type> with:

```
<routine type> ::=
        ROUTINE
      | FUNCTION
      | PROCEDURE
      | [ INSTANCE | STATIC | CONSTRUCTOR ] METHOD
```

3. *Rationale: Allow the specification of a <method name> when a method is being designated.*

In the Format, replace the production of <member name> with:

```
<member name> ::=
        { <schema qualified routine name> | <method name> }
        [ <data type list> ]
```

4. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 2) with:

2) If <routine type> specifies METHOD and neither INSTANCE nor STATIC nor CONSTRUCTOR is specified, then INSTANCE is implicit.

5. *Rationale: Use the correct BNF (<user-defined type name> instead of <user-defined type>) and correct namespace problems associated with methods used to initialize newly-constructed structured type values. Replace incorrectly used concept of data type identity of routine parameters with explicit reference to the Syntax Rules of Subclause 10.14, "Data type identity". Allow the specification of a <method name> when a method is being designated.*

Replace Syntax Rule 3) with:

3) If a <member name> *MN* is specified, then:

   a) If <user-defined type name> is specified, then <routine type> shall specify METHOD. If METHOD is specified, then <user-defined type name> shall be specified.

   b) Case:

i) If <routine type> specifies METHOD, then <method name> shall be specified. Let *SCN* be the implicit or explicit <schema name> of <schema-resolved user-defined type name>, let *METH* be the <method name>, and let *RN* be *SCN.METH*.

ii) Otherwise, <schema qualified routine name> shall be specified. Let *RN* be the <schema qualified routine name> of *MN* and let *SCN* be the <schema name> of *MN*.

c) Case:

i) If *MN* contains a <data type list>, then:

1) If <routine type> specifies FUNCTION, then there shall be exactly one SQL-invoked function that is not an SQL-invoked method in the schema identified by *SCN* whose <schema qualified routine name> is *RN* such that for all *i* the Syntax Rules of Subclause 10.16, ''Data type identity'', when applied with the declared type of its *i*-th SQL parameter and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that SQL-invoked function.

2) If <routine type> specifies PROCEDURE, then there shall be exactly one SQL-invoked procedure in the schema identified by *SCN* whose <schema qualified routine name> is *RN* such that for all *i* the Syntax Rules of Subclause 10.16, ''Data type identity'', when applied with the declared type of its *i*-th SQL parameter and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that SQL-invoked procedure.

3) If <routine type> specifies METHOD, then

Case:

A) If STATIC is specified, then there shall be exactly one static SQL-invoked method of the type identified by <user-defined type name> whose <method name> is *METH* such that for all *i*, the Syntax Rules of Subclause 10.16, ''Data type identity'', when applied with the declared data type of its *i*-th SQL parameter and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that static SQL-invoked method.

B) If CONSTRUCTOR is specified, then there shall be exactly one SQL-invoked constructor method of the type identified by <user-defined type name> whose <method name> is *METH* such that for all *i*, the Syntax Rules of Subclause 10.16, ''Data type identity'', when applied with the declared data type of its *i*-th SQL parameter in the unaugmented <SQL parameter declaration list> and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that SQL-invoked constructor method.

C) Otherwise, there shall be exactly one instance SQL-invoked method of the type identified by <user-defined type name> whose <method name> is *METH* such that for all *i*, the Syntax Rules of Subclause 10.16, ''Data type identity'', when applied with the declared data type of its *i*-th SQL parameter in the unaugmented <SQL parameter declaration list> and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that instance SQL-invoked method.

4) If <routine type> specifies ROUTINE, then there shall be exactly one SQL-invoked routine in the schema identified by *SCN* whose <schema qualified routine name> is *RN* such that for all *i* the Syntax Rules of Subclause 10.16, ''Data type identity'', when applied with the declared type of its *i*-th SQL parameter and the *i*-th <data type> in the <data type list> of *MN*, are satisfied. The <specific routine designator> identifies that SQL-invoked routine.

ii) Otherwise:

1) If <routine type> specifies FUNCTION, then there shall be exactly one SQL-invoked function in the schema identified by *SCN* whose <schema qualified routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked function.

2) If <routine type> specifies PROCEDURE, then there shall be exactly one SQL-invoked procedure in the schema identified by *SCN* whose <schema qualified routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked procedure.

3) If <routine type> specifies METHOD, then

Case:

A) If STATIC is specified, then there shall be exactly one static SQL-invoked method of the user-defined type identified by <user-defined type name> whose <method name> is *METH*. The <specific routine designator> identifies that static SQL-invoked method.

B) If CONSTRUCTOR is specified, then there shall be exactly one SQL-invoked constructor method of the user-defined type identified by <user-defined type name> whose <method name> is *METH*. The <specific routine designator> identifies that SQL-invoked constructor method.

C) Otherwise, there shall be exactly one instance SQL-invoked method of the user-defined type identified by <user-defined type name> whose <method name> is *METH*. The <specific routine designator> identifies that instance SQL-invoked method.

4) If <routine type> specifies ROUTINE, then there shall be exactly one SQL-invoked routine in the schema identified by *SCN* whose <schema qualified routine name> is *RN*. The <specific routine designator> identifies that SQL-invoked routine.

6. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 4) with:

4) If FUNCTION is specified, then the SQL-invoked routine that is identified shall be an SQL-invoked function that is not an SQL-invoked method. If PROCEDURE is specified, then the SQL-invoked routine that is identified shall be an SQL-invoked procedure. If STATIC METHOD is specified, then the SQL-invoked routine that is identified shall be a static SQL-invoked method. If CONSTRUCTOR METHOD is specified, then the SQL-invoked routine shall be an SQL-invoked constructor method. If INSTANCE METHOD is specified or implicit, then the SQL-invoked routine shall be an instance SQL-invoked method. If ROUTINE is specified, then the SQL-invoked routine that is identified is either an SQL-invoked function or an SQL-invoked procedure.

10.8 <collate clause>

1. Rationale: Standardise terminology.

Replace the Function with:

Specify a default collation.

10.12 Execution of triggers

1. Rationale: Address requirement for multiple diagnostics areas

Insert the following General Rule:

4) a.0) The General Rules of Subclause 19.2, "Pushing and popping diagnostics areas" are applied, with PUSH as *OPERATION* and the diagnostics area stack as *STACK*.

2. Rationale: Editorial.

Replace General Rule 4) b) with:

4) b) If the execution of *TSS* is not successful, then an exception condition is raised: *triggered action exception*. The exception information associated with *TSS* is entered into the diagnostics area in a location other than the location corresponding to condition number 1 (one).

3. Rationale: Address requirement for multiple diagnostics areas

Insert the following General Rule:

4) c.1) The General Rules of Subclause 19.2, "Pushing and popping diagnostics areas" are applied, with POP as *OPERATION* and the diagnostics area stack as *STACK*.

10.13 Execution of array-returning functions

1. Rationale: Editorial.

Replace General Rule 6) with:

6) If the call type data item has a value of -1 (indicating "*open call*"), then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

a) Depending on whether the language of *R* specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, let the *operative data type correspondences* table be Table 18, "Data type correspondences for Ada", Table 19, "Data type correspondences for C", Table 20, "Data type correspondences for COBOL", Table 21, "Data type correspondences for Fortran", Table 22, "Data type correspondences for MUMPS", Table 23, "Data type correspondences for Pascal", or Table 24, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the "SQL data type" column and the "host data type" column.

b) For *i* varying from 1 (one) to *EN*, the <data type> DT_i of PD_i is the data type listed in the host data type column of the row in the data type correspondences table whose value in the SQL data type column corresponds to the data type of ESP_i.

c) The value of PD_i is set to the value of ESP_i.

Replace General Rule 8) a) with:

8) a) *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

i) Depending on whether the language of *R* specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, let the *operative data type correspondences* table be Table 18, "Data type correspondences for Ada", Table 19, "Data type correspondences for C", Table 20, "Data type correspondences for COBOL", Table 21, "Data type correspondences for Fortran", Table 22, "Data type correspondences for MUMPS", Table 23, "Data type correspondences for Pascal", or Table 24, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the "SQL data type" column and the "host data type" column.

ii) For *i* varying from 1 (one) to *EN*, the <data type> DT_i of PD_i is the data type listed in the host data type column of the row in the data type correspondences table whose value in the SQL data type column corresponds to the data type of ESP_i.

c) The value of PD_i is set to the value of ESP_i.

Replace General Rule 8) b) iii) with:

8) b) iii) Otherwise, set the value of the call type data item to 1 (one) (indicating *close call*).

Replace General Rule 9) with:

9) If the call type data item has a value of 1 (indicating "*close call*"), then *P* is executed with a list of *EN* parameters PD_i whose parameter names are PN_i and whose values are set as follows:

a) Depending on whether the language of *R* specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, let the *operative data type correspondences* table be Table 18, "Data type correspondences for Ada", Table 19, "Data type correspondences for C", Table 20, "Data type correspondences for COBOL", Table 21, "Data type correspondences for Fortran", Table 22, "Data type correspondences for MUMPS", Table 23, "Data type correspondences for Pascal", or Table 24, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the "SQL data type" column and the "host data type" column.

b) For *i* varying from 1 (one) to *EN*, the <data type> DT_i of PD_i is the data type listed in the host data type column of the row in the data type correspondences table whose value in the SQL data type column corresponds to the data type of ESP_i.

c) The value of PD_i is set to the value of ESP_i.

10.14 Data type identity

1. Rationale: The requirement that the data type identity should subsume data type compatibility is not enforced.

Replace all the Syntax Rules of this Subclause with the following:

1) Let *PM* and *P* be the two data types specified in an application of this Subclause.

2) *PM* and *P* shall be compatible.

3) If *PM* is a character string type, then the length of *PM* shall be equal to the length of *P*.

4) If *PM* is an exact numeric type, then the precision and scale of *PM* shall be equal to the precision and scale *P*, respectively.

5) If *PM* is an approximate numeric type, then the precision of *PM* shall be equal to the precision of *P*.

6) If *PM* is binary string type, then the maximum length of *PM* shall be equal to the maximum length of *P*.

7) If *PM* is a bit string type, then the length of *PM* shall be equal to the length of *P*.

8) If *PM* is a datetime data type with <time fractional seconds precision>, then the <time fractional seconds precision> of *PM* shall be equal to the <time fractional seconds precision> of *P*.

9) If *PM* is an interval type, then the <interval qualifier> of *PM* shall be equivalent to the <interval qualifier> of *P*.

10) If *PM* is a collection type, then

    a) The maximum cardinality of *PM* shall be equal to the maximum cardinality of *P*.

    b) The Syntax Rules of this Subclause are applied with the element type of *PM* and the element type of *P* as the two data types.

11) If *PM* is a row type, then:

    a) Let *N* be the degree of *PM*.

    b) Let $DTFPM_i$ and $DTFP_i$ be the data type of *i*-th field of *PM* and *P*, respectively. For *i* varying from 1 (one) to *N*, the Syntax Rules of this Subclause are applied with $DTFPM_i$ and $DTFP_i$ as the two data types.

11.1 <schema definition>

1) Rationale: Remove redundant item. <grant role statement> is defined in <grant statement>.

In the Format, replace the production for <schema element> with:

```
<schema element> ::=
       <table definition>
     | <view definition>
     | <domain definition>
     | <character set definition>
     | <collation definition>
     | <translation definition>
     | <assertion definition>
     | <trigger definition>
     | <user-defined type definition>
     | <schema routine>
     | <grant statement>
     | <role definition>
     | <user-defined cast definition>
     | <user-defined ordering definition>
     | <transform definition>
```

2) Rationale: Correct the references to schema which do not treat it as a descriptor.

Replace General Rules 1) with:

1) A <schema definition> creates an SQL-schema *S* in a catalog. *S* includes:

   a) A schema name that is equivalent to the explicit or implicit <schema name>.

   b) A schema authorization identifier that is equivalent to the explicit or implicit <authorization identifier>.

   c) A schema character set name that is equivalent to the explicit or implicit <schema character set specification>.

   d) A schema SQL-path that is equivalent to the explicit or implicit <schema path specification>.

   e) The descriptor created by every <schema element> of the <schema definition>.

Delete General Rule 4).

3) Rationale: Remove redundant and misleading Conformance Rules.

Delete Conformance Rules 1), 2), 4), 5), 6), 7), 9) and 11).

4. *Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".*

Replace Conformance Rules 8) with:

8) Without Feature F461, "Named character sets", a <schema character set specification> shall not be specified.

11.2 <drop schema statement>

1. *Rationale: DROP ASSERTION statement now requires CASCADE.*

Replace General Rule 4) with:

4) Let *A* be the <constraint name> included in the descriptor of any assertion included in *S*. The following <drop assertion statement> is effectively executed:

```
DROP ASSERTION A CASCADE
```

2. *Rationale: Standardise terminology.*

Replace General Rule 5) with:

5) Let *CD* be the <collation name> included in the descriptor of any collation included in *S*. The following <drop collation statement> is effectively executed:

DROP COLLATION *CD* CASCADE

3. *Rationale: It is impossible for an implementation to know all SQL routine bodies that generally contain a <schema name>.*

Replace General Rule 11) with:

11) Let *R* be any SQL-invoked routine whose routine descriptor includes an SQL routine body that contains the <schema name> of *S*. Let *SN* be the <specific name> of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:

DROP SPECIFIC ROUTINE *SN* CASCADE

4. *Rationale: Correct the references to schema which do not treat it as a descriptor.*

Replace General Rule 13) with:

13) *S* is destroyed.

11.3 <table definition>

1. *Rationale: Incorrect symbol in syntax substitute for <like clause>.*

Replace Syntax Rule 6) b) with:

6) b) Let *nt* be the number of columns in *T1*. Let C_i, 1 (one) ≤ i ≤ *nt*, be the columns of *T1*, in the order in which they appear in *T1*, let CN_i be the column name included in the column descriptor of C_i, and let DT_i be the data type included in the column descriptor of C_i. Let CD_1 be:

CN_1 DT_1

If *nt* is greater than 1 (one), then let CD_i, 2 ≤ i ≤ *nt*, be:

, CN_i DT_i

The <like clause> is effectively replaced by CD_i, 1 (one) ≤ i ≤ *nt*.
NOTE 169 – <column constraint>s are not included in columns; <column constraint>s are effectively transformed to <table constraint>s and are thereby excluded.

2. *Rationale: Recognise that unique constraints may contain more than one column*

Replace Syntax Rule 7) g) with:

7) g There shall exist some supertable of *T* whose table descriptor includes a unique constraint descriptor *UCD* such that the nullability characteristic included in the column descriptor of every column whose column name is included in *UCD* is *known not nullable*.

3. *Rationale: Remove the injudicious option to specify a column to have a collation different from that of the corresponding attribute .*

In the Format, replace the production for <column option list> with:

```
<column option list> ::=
  [ <scope clause> ]
  [ <default clause> ]
  [ <column constraint definition>... ]
```

Replace Syntax Rule 9) with:

9) For every <column options> *CO*, <column name> shall be equivalent to the <column name> specified in some <column definition> *RCD* implicitly or explicitly contained in *TD* and shall not refer to an inherited column of *T*. Distinct <column options>s contained in *TD* shall specify distinct <column name>s.

a) If *CO* specifies a <scope clause> *SC*, then let *CURITIBA* be the <column name> contained in *RCD* followed in turn by the <data type> or <domain name> contained in *RCD*, *SC*, the <default clause> (if any) contained in *RCD*, and every <column constraint definition> contained in *RCD*. *RCD* is replaced by *CURITIBA*.

b) If *CO* specifies <default clause> *DC*, then let *CURITIBA* be the <column name> contained in *RCD* followed in turn by the <data type> or <domain name> contained in *RCD*, *DC*, and every <column constraint definition> contained in *RCD*. *RCD* is replaced by *CURITIBA*.

c) If *CO* specifies a non-empty list *CCDL* of <column constraint definition>s, then let *CURITIBA* be the <column name> contained in *RCD* followed in turn by the <data type> or <domain name> contained in *RCD*, and the <default clause> (if any) contained in *RCD*, *CCDL*. *RCD* is replaced by *CURITIBA*.

4. *Rationale: Incomplete rule—should prohibit <column definition>s as well as <like clause>s and replace the use of incorrect tags.*

Replace Syntax Rule 10) a) with:

10) a) The <user-defined type name> simply contained in <user-defined type> shall identify a structured type *ST*. Let the <table element list>, if specified, be *TEL*.

Replace Syntax Rule 10) b) with:

10) b) *TEL* shall not contain a <like clause> or a <column definition>.

5. *Rationale: Replace incorrect tags and other text.*

Replace Syntax Rule 10) d) ii) with:

10) d) ''OF <user-defined type>'' is effectively replaced by a <table element list> *TEL1 TEL2*, where *TEL1* and *TEL2* are defined as follows:

TEL1 consists of n <table element>s, where n is the number of attribute descriptors included in the data type descriptor of *ST*. For each attribute descriptor *AD* included in the data type descriptor of *ST*, the corresponding <table element> in *TEL1* is the <column definition> *CN DT DC CC*, where:

i) *CN* is the attribute name included in *AD*.

ii) *DT* is some <data type> that, under the General Rules of Subclause 6.1, ''<data type>'', would result in the creation of the data type descriptor included in *AD*.

iii) Case:

1) If *AD* describes an inherited attribute *IA*, then *DC* is some <default clause> whose <default option> denotes the default value included in the column descriptor of the direct supercolumn of *IA*.

2) Otherwise, *DC* is some <default clause> whose <default option> denotes the default value included in *AD*.

iv) Case:

1) If *AD* describes an inherited attribute *IA*, and the descriptor of the direct supercolumn of *IA* includes a <collation name> *COLIN*, then *CC* is ''COLLATE *COLIN*''.

2) If *AD* describes an inherited attribute *IA*, and the descriptor of the direct supercolumn of *IA* does not include a <collation name>, then *CC* is a zero-length string.

3) If *AD* includes a <collation name> *COLIN*, then *CC* is ''COLLATE *COLIN*''.

4) Otherwise *CC* is a zero-length string.

If <table element list> *TEL* is specified and contains a <table element> that is not a <column definition>, then *TEL2* is a <comma> followed by those <table elements> of *TEL* that are not <column definition>s, the members of each adjacent pair being separated by a <comma>; otherwise *TEL2* is a zero-length string.

6. *Rationale: Align the rules with the BNF in the Format,*

Replace Syntax Rule 16) with:

16) If *TEL1* contains a <column options>, then *TD* shall specify OF <user-defined type> and any <column definition> shall be contained before the first <column options>.

7. *Rationale: Align the rules with the BNF in the Format.*

Replace Access Rule 4) with:

4) If "OF <user-defined type>" is specified, then the applicable privileges of *A* shall include USAGE on *ST*.

8. *Rationale: Editorial - typographical error.*

Replace General Rule 3) with:

3) For each <column options> CO, if CO contains a <scope clause> SC, then let CD be the column descriptor identified by the <column name> specified in CO. The <table name> specified in SC is included in the reference type descriptor that is included in CD.

9. *Rationale: Correct the default of <table commit action>.*

Replace General Rule 5) l) with:

5) l) If TEMPORARY is specified, then

Case:

i) If ON COMMIT PRESERVE ROWS is specified, then the table descriptor includes an indication that ON COMMIT PRESERVE ROWS is specified.

ii) Otherwise, the table descriptor includes an indication that ON COMMIT DELETE ROWS is specified or implied.

10. *Rationale: Clean up typed table insertability property for non-instantiable types.*

Insert the following General Rule:

5) l.1) Case:

i) If OF <user-defined type> is not specified, then an indication that *T* is insertable-into.

ii) Otherwise,

Case:

1) If the data type descriptor of *R* indicates that *R* is instantiable, then an indication that *T* is insertable-into.

2) Otherwise, an indication that *T* is not insertable-into.

11. Rationale: Correct privileges for <temporary tables>s.

Replace General Rule 8) with:

8) A set of privilege descriptors is created that define the privileges INSERT, SELECT, UPDATE, DELETE, TRIGGER, and REFERENCES on this table and SELECT, INSERT, UPDATE, and REFERENCES for every <column definition> in the table definition. If OF <user-defined type> is specified, then a table/method privilege descriptor is created on this table for every method of the structured type identified by the <user-defined type> and the table SELECT privilege has the WITH HIERARCHY OPTION. These privileges are grantable. The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM". The grantee is <authorization identifier> *A*.

12. Rationale: Editorial.

Replace General Rule 10) with:

10) The row type *RT* of the table *T* defined by the <table definition> is the set of pairs (<field name>, <data type>) where <field name> is the name of a column *C* of *T* and <data type> is the declared type of *C*. This set of pairs contains one pair for each column of *T*, in the order of their ordinal position in *T*.

13. Rationale: Permit tables of structured type without requiring Feature S043, "Enhanced reference types".

Replace Conformance Rule 5) with:

5) Without Feature S043, "Enhanced reference types", a <self-referencing column specification> shall specify SYSTEM GENERATED.

11.4 <column definition>

1. Rationale: Provide consistent Syntax Rules for comparison operations.

Insert the following Syntax Rule:

9.1) If <data type> is a <reference type> that identifies a reference type that is LOB-ordered, array-ordered, UDT-EC-ordered or UDT-NC-ordered, then REFERENCES ARE CHECKED shall not be specified.

2. Rationale: Syntax Rule 13) should not apply to <array specification>.

Replace Syntax Rule 13) with:

13) If <data type> simply contains a <row type>, or a <path-resolved user-defined type name> that identifies a user-defined type descriptor whose degree is greater than 0 (zero), then let *CDTD* be the descriptor associated with that <row type>, or <path-resolved user-defined type name>, respectively.

3. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Insert the following Conformance Rule:

7) Without Feature S024, "Enhanced structured types", if <data type> is a <reference type> that identifies a reference type that is ST-ordered, then REFERENCES ARE CHECKED shall not be specified.

4. *Rationale: Add a missing (but inferable) Conformance Rules.*

Insert the following Conformance Rule:

10) Without Feature S041, "Basic reference types", conforming SQL language shall not contain any <reference scope check>.

11.5 <default clause>

1. *Rationale: Ensure that only appropriate datetime literals can be used as defaults.*

Replace Syntax Rule 4)a)vi) with:

4) a) vi) If the subject data type is datetime, then the <literal> shall be a <datetime literal> with the same primary datetime fields and the same time zone datetime fields as the subject data type. If SECOND is one of these fields, then the fractional seconds precision of the <datetime literal> shall be less than or equal to the fractional seconds precision of the subject data type.

2. *Rationale: Remove redundant rule and relocate note.*

Replace Conformance Rules 2) and 3) with:

3) Without Feature F321, ''User authorization'', a <default option> shall not be CURRENT_USER, SESSION_USER, or SYSTEM_USER.
NOTE 179 – Although CURRENT_USER and USER are semantically the same, in Core SQL, CURRENT_USER must be specified as USER.

11.7 <unique constraint definition>

1. *Rationale: Provide consistent Syntax Rules for comparison operations.*

Replace Syntax Rule 1) with:

1) The declared type of no column identified by any <column name> in the <unique column list> shall be LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

2. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Insert the following Conformance Rule:

3) Without Feature S024, "Enhanced structured types", the declared type of no column identified by any <column name> in the <unique column list> shall be of ST-ordered declared type.

11.8 <referential constraint definition>

1. *Rationale: Add a missing prefix.*

Replace Syntax Rule 3) b) ii) with:

3) b) ii) For a given row in the referenced table, every matching row for that given row that is a matching row only to the given row in the referenced table for the referential constraint is a *unique matching row*. For a given row in the referenced table, a matching row for that given row that is not a unique matching row for that given row for the referential constraint is a *non-unique matching row*.

2. *Rationale: Provide consistent Syntax Rules for comparison operations.*

Insert the following Syntax Rule:

8.1) No referencing column shall have a declared type that is LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

3. *Rationale: referential constraints should not depend on a possibly non-deterministic comparison.*

Insert the following Syntax Rule:

8.2) There shall not be corresponding constituents of the declared type of a referencing column and the declared type of the corresponding referenced column such that one constituent is datetime with time zone and the other is datetime without time zone.

4. *Rationale: Provide the rule of initialization of transition table and association between subject table of trigger and transition table on each trigger context.*

Insert the following General Rule:

6.1) If no SC_j in *SSC* has *F* as subject table, then

Case:

a) If <delete rule> specifies CASCADE, then old transition table is initialized as empty table and associated with *F* on *CTEC*.

b) If <delete rule> specifies SET NULL or SET DEFAULT, or if <update rule> specifies CASCADE, SET NULL or SET DEFAULT, then old transition table and new transition table are initialized as empty table and associated with *F* on *CTEC*.

5. *Rationale: Provide the rule of construction of the value of transition variable.*

Insert the following General Rule:

7) a) i) 0) For each matching row, the value of its old transition variable is constructed by copying the matching row.

6. *Rationale: Provide the rule of construction of the value of transition variable.*

Insert the following General Rule:

7) a) ii) 0) For every *F*, for each matching row in *F*, the value of its old transition variable is constructed by copying the matching row, and the value of new transition variable is constructed by copying the matching row and setting each referencing column in the row being constructed to the null value.

7. *Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.*

Replace General Rule 7) a) ii) 2) with:

7) a) ii) 2) For every matching row in every *F*:

A) The matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

8. *Rationale: Provide the rule of construction of the value of transition variable.*

Insert the following General Rule:

7) a) iii) 0) For every *F*, for each matching row in *F*, the value of its old transition variable is constructed by copying the matching row, and the value of new transition variable is constructed by copying the matching row and setting each referencing column in the row being constructed to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

9. *Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.*

Replace General Rule 7) a) iii) 2) with:

7) a) iii) 2) For every matching row in every *F*:

A) The matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

10. *Rationale: Provide the rule of construction of the value of transition variable.*

Insert the following General Rule:

7) b) i) 0) For each unique matching row, the value of its old transition variable is constructed by copying the unique matching row.

11. Rationale: Provide the rule of construction of the value of transition variable.

Insert the following General Rule:

7) b) ii) 0) For every *F*, for each unique matching row in *F*, the value of its old transition variable is constructed by copying the unique matching row, and the value of new transition variable is constructed by copying the unique matching row and setting each referencing column in the row being constructed to the null value.

12. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 7) b) ii) 2)with:

7) b) ii) 2) For every unique matching row in every *F*:

A) The unique matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

13. Rationale: Provide the rule of construction of the value of transition variable.

Insert the following General Rule:

7) b) iii) 0) For every *F*, for each unique matching row in *F*, the value of its old transition variable is constructed by copying the unique matching row, and the value of new transition variable is constructed by copying the unique matching row and setting each referencing column in the row being constructed to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

14. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 7) b) iii) 2) with:

7) b) iii) 2) For every unique matching row in every *F*:

A) The unique matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

15. Rationale: Provide the rule of construction of the value of transition variable.

Insert the following General Rule:

8) a) i) 0) For every *F*, for each matching row in *F*, the value of its old transition variable is constructed by copying the matching row, and the value of new transition variable is constructed by copying the matching row and updating each referencing column in the row

being constructed that is corresponds with a referenced column to new value of that referenced column.

16. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 8) a) i) 2) with:

8) a) i) 2) For every matching row in every *F*:

A) The matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

17. Rationale: Provide the rule of construction of the value of transition variable.

Insert the following General Rule:

8) a) ii) 1) A.0) For every *F*, for each matching row in *F*, the value of its old transition variable is constructed by copying the matching row, and the value of new transition variable is constructed by copying the matching row and setting each referencing column in the row being constructed to the null value.

18. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 8) a) ii) 1) B) with:

8) a) ii) 1) B) For every matching row in every *F*:

I) The matching row is replaced by its new transition variable.

II) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

19. Rationale: Provide the rule of construction of the value of transition variable.

Insert the following General Rule:

8) a) ii) 2) A.0) For every *F*, for each matching row in *F*, the value of its old transition variable is constructed by copying the matching row, and the value of new transition variable is constructed by copying the matching row and setting each referencing column in the row being constructed that is corresponds with a referenced column to the null value.

20. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 8) a) ii) 2) B) with:

8) a) ii) 2) B) For every matching row in every *F*:

I) The matching row is replaced by its new transition variable.

II) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

21. Rationale: A foreign key can consist of several columns. Provide the rule of construction of the value of transition variable.

Insert the following General Rule:

8) a) iii) 0) For every *F*, for each matching row in *F*, the value of its old transition variable is constructed by copying the matching row, and the value of new transition variable is constructed by copying the matching row and setting each referencing column in the row being constructed that is corresponds with a referenced column to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

22. Rationale: A foreign key can consist of several columns. Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 8) a) iii) 2) with:

8) a) iii) 2) For every matching row in every *F*:

A) The matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

23. Rationale: Provide the rule of construction of the value of transition variable.

Insert the following General Rule:

8) b) i) 1) 0) For every *F*, for each unique matching row in *F* that contains a non-null value in the referencing column *C1* in *F* that corresponds with the updated referenced column *C2*, the value of its old transition variable is constructed by copying the matching row, and the value of new transition variable is constructed by copying the matching row and updating *C1* in the row being constructed to the new value *V* of *C2*, provided that, in all updated rows in the referenced table that formerly had, in the same SQL-statement, that unique matching row as a matching row, the values in *C2* have all been updated to a value that is not distinct from *V*. Otherwise, an exception condition is raised: *triggered data change violation*.
NOTE 253 – Because of the Rules of Subclause 8.2, ''<comparison predicate>'', on which the definition of ''distinct'' relies, the values in *C2* may have been updated to values that are not distinct, yet are not identical. Which of these non-distinct values is used for the cascade operation is implementation-dependent.

24. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 8) b) i) 2) with:

8) b) i) 2) For every unique matching row in every *F*:

A) The unique matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

25. Rationale: Provide the rule of construction of the value of transition variable.

Insert the following General Rule:

8) b) ii) 0) For every *F*, for each unique matching row in *F* that contains a non-null value in the referencing column in *F* that corresponds with the updated referenced column, the value of its old transition variable is constructed by copying the unique matching row, and the value of new transition variable is constructed by copying the matching row and setting that referencing column in the row being constructed to the null value.

26. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 8) b) ii) 2) with:

8) b) ii) 2) For every unique matching row in every *F*:

A) The unique matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

27. Rationale: Provide the rule of construction of the value of transition variable.

Add following General Rule:

8) b) iii) 0) For every *F*, for each unique matching row in *F* that contains a non-null value in the referencing column in *F* that corresponds with the updated referenced column, the value of its old transition variable is constructed by copying the unique matching row, and the value of new transition variable is constructed by copying the matching row and setting that referencing column in the row being constructed to the default value specified in the General Rules of Subclause 11.5, "<default clause>".

28. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 8) b) iii) 2) with:

8) b) iii) 2) For every unique matching row in every *F*:

A) The unique matching row is replaced by its new transition variable.

B) The old transition variable is inserted into old transition table associated with *F* on *CTEC* and the new transition variable is inserted into new transition table associated with *F* on *CTEC*.

29. Rationale: Provide the rule of insertion of the value of transition variable.

Add following General Rule:

13.1) For all rows that are marked for deletion, the old transition variable is inserted into old transition table associated with *F* on *CTEC*.

30. Rationale: Provide consistent Conformance Rules for comparison operations.

Insert the following Conformance Rule:

6) Without Feature S024, "Enhanced structured types", no referencing column shall be of ST-ordered declared type.

11.9 <check constraint definition>

1. Rationale: Use the correct definition of possibly non-deterministic.

Replace Syntax Rules 5), 6) and 7) with:

5) The <search condition> shall not generally contain a <value expression>, <query specification> or <query expression> that is possibly non-deterministic.

11.16 <drop column scope clause>

1. Rationale: Remove redundant and confusing references to assertion descriptor.

Replace Syntax Rule 3) c) as follows:

3) c) The <search condition> of any constraint descriptor.

2. Rationale: DROP ASSERTION statement now requires CASCADE.

Replace General Rule 3) with:

3) For every assertion *A* whose assertion descriptor includes a <search condition> that contains an impacted dereference operation, let *AN* be the <constraint name> of *A*. The following <drop assertion statement> is effectively executed for every *A* without further Access Rule checking:

```
DROP ASSERTION AN CASCADE
```

11.17 <drop column definition>

1. Rationale: Remove unneeded misleading text.

Delete NOTE 193

Delete General Rule 2).

11.19 <drop table constraint definition>

1. Rationale: A <query specification> need not have a <grouping column reference list>, but it always has a set of grouping columns (possibly empty). Capture all the dependencies on functional dependencies deduced from a table constraint.

Replace Syntax Rule 4) with:

4) If *QS* is a <query specification> that contains an implicit or explicit <group by clause> and that contains a column reference to a column *C* in its <select list> that is not contained in a <set function specification>, and if *G* is the set of grouping columns of *QS*, and if the table constraint *TC* is needed to conclude that $G \rightarrow C$ is a known functional dependency in *QS*, then *QS* is said to be *dependent on TC*.

4.1) If *V* is a view that contains a <query specification> that is dependent on a table constraint *TC*, then *V* is said to be *dependent on TC*.

4.2) If *R* is an SQL routine whose <SQL routine body> contains a <query specification> that is dependent on a table constraint *TC*, then *R* is said to be *dependent on TC*.

4.3) If *C* is a constraint or assertion whose <search condition> contains a <query specification> that is dependent on a table constraint *TC*, then *C* is said to be *dependent on TC*.

4.4) If *T* is a trigger whose triggered action contains a <query specification> that is dependent on a table constraint *TC*, then *T* is said to be *dependent on TC*.

Insert the following Syntax Rules:

6) d) No SQL-routine shall be dependent on *TC*

e) No constraint or assertion shall be dependent on *TC*

f) No trigger shall be dependent on *TC*.

Insert the following General Rules:

3.1) Let *SN* be the specific name of any SQL routine *SR* that is dependent on *TC*. The following <drop routine statement> is effectively executed for every *SR*:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

3.2) Let *CN* be the constraint name of any constraint *C* that is dependent on *TC*. Let *TN* be the name of the table constrained by *C*. The following <alter table statement> is effectively executed for every *C*:

```
ALTER TABLE TN DROP CONSTRAINT CN CASCADE
```

3.3) Let *AN* be the assertion name of any assertion *A* that is dependent on *TC*. The following <drop assertion statement> is effectively executed for every *A*:

```
DROP ASSERTION VN CASCADE
```

3.4) Let *TN* be the trigger name of any trigger *T* that is dependent on *TC*. The following <drop trigger statement> is effectively executed for every *T*:

```
DROP TRIGGER TN
```

11.20 <drop table statement>

1. Rationale: Remove redundant and confusing references to assertion descriptor.

Replace Syntax Rule 6) b) as follows:

6) b) The <search condition> of any constraint descriptor that is not a table check constraint descriptor included in the base table descriptor of *T*.

11.21 <view definition>

1. Rationale: permit <self-referencing column specification> as the only option in <view element list>.

Replace the Format for <view element list> with:

```
<view element list> ::=
     <left paren>
     <view element> [ { <comma> <view element> }... ]
     <right paren>
```

2. Rationale: permit <self-referencing column specification> as the only option in <view element list>.

Replace the Format for <view element> with:

```
<view element> ::=
       <self-referencing column specification>
     | <view column option>
```

3. Rationale: Correct syntax of syntactic substitution.

Replace Syntax Rule 8) e) with:

8) e) *VD* is equivalent to

```
CREATE VIEW <table name> AS
  WITH RECURSIVE <table name> (<view column list>)
    AS (<query expression>)
  SELECT <view column list> FROM <table name>
```

4. Rationale: permit <self-referencing column specification> as the only option in <view element list>.

Insert the following Syntax Rule:

20.1) <view element list> shall specify at most one <self-referencing column specification>.

5. Rationale: Replace incorrect non-terminal.

Replace Syntax Rule 21) e) with:

21) e) If <subview clause> is not specified, then <self-referencing column specification> shall be specified.

Replace Syntax Rule 21) i) with:

21) i) If <self-referencing column specification> is specified, then:

i) <subview clause> shall not be specified.

ii) SYSTEM GENERATED shall not be specified.

iii) Let *RST* be the reference type REF(*ST*).

Case:

1) If USER GENERATED is specified, then:

A) *RST* shall have a user-defined representation.

B) Let *m* be 1 (one).

2) If DERIVED is specified, then:

A) *RST* shall have a derived representation.

B) Let *m* be 0 (zero).

6. Rationale: Replace incorrect symbol.

Replace Syntax Rule 21) j) iv) with:

21) j) iv) Let *MSV* be the maximum superview of the subtable family of *V*. Let *RMSV* be the reference type REF(*MSV*).

Case:

1) If *RMSV* has a user-defined representation, then let *m* be 1 (one).

2) Otherwise, *RMSV* has a derived representation. Let *m* be 0 (zero).

7. *Rationale: Replace incorrect non-terminal.*

Replace Syntax Rule 21) s) with:

21) s) If <self-referencing column specification> is specified, then

Case:

i) If *RST* has a user-defined representation, then:

1) *TQN* shall have a candidate key consisting of a single column *RC*.

2) Let *SS* be the first <select sublist> in the <select list> of *QS*.

3) *SS* shall consist of a single <cast specification> *CS* whose leaf column is *RC*.

NOTE 202 — "Leaf column of a <cast specification>" is defined in Subclause 6.22, "<cast specification>".

4) The declared type of F_1 shall be REF(*ST*).

ii) Otherwise, *RST* has a derived representation.

1) Let C_i, 1 (one) ≤ i ≤ n, be the columns of *V* that correspond to the attributes of the derived representation of *RST*.

2) *TQN* shall have a candidate key consisting of some subset of the underlying columns of C_i, 1 (one) ≤ i ≤ n.

11.22 <drop view statement>

1. *Rationale: Remove redundant and confusing references to assertion descriptor.*

Replace Syntax Rule 4) c) as follows:

4) b) The <search condition> of any constraint descriptor.

11.23 <domain definition>

1. *Rationale: Exclude row types from domains.*

Replace Syntax Rule 6) with:

6) <data type> shall be <predefined type>.

11.30 <character set definition>

1. *Rationale: Replace undefined non-terminals .*

Replace Syntax Rule 3) with:

3) The character set *CS* identified by the <character set specification> contained in <character set source> shall have associated with it a privilege descriptor that was effectively defined by the <grant statement>

```
GRANT USAGE ON CHARACTER SET CSN TO PUBLIC
```

where *CSN* is a <character set name> that identifies *CS*.

2. *Rationale: Insist on applicability of the default collation.*

Replace Syntax Rule 4) with:

4) If <collate clause> is specified, then it shall contain a <collation name> that identifies a collation descriptor *CD* included in the schema identified by the explicit or implicit <schema name> contained in the <collation name>. The list of applicable character set names included in *CD* shall include one that identifies *CS*.

3. *Rationale: Delete unnecessary and incorrect rule .*

Delete General Rule 4).

4. *Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets" and remove redundant rule .*

Replace Conformance Rules 1) and 2) with:

1) Without Feature F451, “Character set definition”, conforming SQL language shall not specify any <character set definition>.

11.31 <drop character set statement>

1. *Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".*

Replace Conformance Rule 1) with:

1) Without Feature F451, “Character set definition”, conforming SQL language shall contain no <drop character set statement>.

11.32 <collation definition>

1. Rationale: Standardise terminology.

Replace the Function with:

Define a collation.

Replace Syntax Rule 6) with:

6) The collation identified by *ECN* shall be a collation whose descriptor includes a character repertoire name that is equivalent to that included in the descriptor of the character set identified by <character set specification>.

Replace General Rule 1) with:

1) A <collation definition> defines a collation.

11.33 <drop collation statement>

1. Rationale: Standardise terminology.

Replace the Function with:

Destroy a collation.

Replace Syntax Rules 1) and 5) with:

1) Let *C* be the collation identified by the <collation name> and let *CN* be the name of *C*.

5) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the collation identified by *C*.

Replace General Rules 1) and 2) with:

2) For every character set descriptor *CSD* that includes *CN*, *CSD* is modified such that it does not include *CN*. If *CSD* does not include any transliteration name, then *CSD* is modified to indicate that it utilizes the default collation for its character encoding form.

11.35 <translation definition>

1. Rationale: Clarify the distinction between character sets and character repertoires.

Replace Syntax Rule 5) with:

5) If <existing transliteration name> is specified, then:

a) The schema identified by the explicit or implicit schema name of the <transliteration name> *TN* contained in <transliteration source> shall include a transliteration descriptor whose transliteration name is *TN*.

b) The character set identified by *SCSN* shall have the same character repertoire and character encoding form as the source character set of the transliteration identified by *TN*.

c) The character set identified by *TCSN* shall have the same character repertoire and character encoding form as the target character set of the transliteration identified by *TN*.

Replace General Rule 2) with:

2) A transliteration descriptor is created including the following:

— The name of the transliteration, *TN*

— The name of the character set, *SCSN*, from which it translates

— The name of the character set, *TCSN*, to which it translates

— An indication of how the transliteration is performed.

11.36 <assertion definition>

1. Rationale: Use the correct definition of possibly non-deterministic.

Replace Syntax Rules 6) and 7) with:

6) The <search condition> shall not generally contain a <value expression>, <query specification> or <query expression> that is possibly non-deterministic.

Delete Syntax Rule 10).

11.37 <drop assertion statement>

1. Rationale: capture all the dependencies on functional dependencies deduced from an assertion; also, an assertion cannot be referenced in a triggered action.

Replace the Format with the following:

```
<drop assertion statement> ::=
    DROP ASSERTION <constraint name> [ <drop behavior> ]
```

Replace Syntax Rule 3) with:

3) If <drop behavior> is not specified, then RESTRICT is implicit.

3.1) If RESTRICT is specified or implied, then *AN* shall not be referenced in the SQL routine body of any routine descriptor.

3.2) If *QS* is a <query specification> that contains a column reference to a column *C* in its <select list> that is not contained in a <set function specification>, and if *G* is the set of columns defined by the <grouping column reference list> of *QS*, and if the assertion *A* is needed to conclude that $G \rightarrow C$ is a known functional dependency in *QS*, then *QS* is said to be *dependent on A*.

3.3) If *V* is a view that contains a <query specification> that is dependent on *A*, then *V* is said to be *dependent on A*.

3.4) If *R* is an SQL routine whose <SQL routine body> contains a <query specification> that is dependent on *A*, then *R* is said to be *dependent on A*.

3.5) If *C* is a constraint or assertion whose <search condition> contains a <query specification> that is dependent on *A*, then *C* is said to be *dependent on A*.

3.6) If *T* is a trigger whose triggered action contains a <query specification> that is dependent on *A*, then *T* is said to be *dependent on A*.

3.7) If RESTRICT is specified or implicit, or <drop behavior> is not specified, then:

a) No table constraint shall be dependent on *A*.

b) No view shall be dependent on *TC*.

c) No SQL routine shall be dependent on *TC*

d) No constraint or assertion shall be dependent on *TC*

e) No trigger shall be dependent on *TC*.

Replace General Rule 2) with:

2.1) Let *VN* be the table name of any view *V* that is dependent on *A*. The following <drop view statement> is effectively executed for every *V*:

```
DROP VIEW VN CASCADE
```

2.2) Let *SN* be the specific name of any SQL routine *SR* that is dependent on *A*, or that contains a reference to *A*. The following <drop routine statement> is effectively executed for every *SR*:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

2.3) Let *CN* be the constraint name of any constraint *C* that is dependent on *A*. Let *TN* be the name of the table constrained by *C*. The following <alter table statement> is effectively executed for every *C*:

```
ALTER TABLE TN DROP CONSTRAINT CN CASCADE
```

2.4) Let *AN2* be the assertion name of any assertion *A* that is dependent on *A*. The following <drop assertion statement> is effectively executed for every *A*:

```
DROP ASSERTION AN2 CASCADE
```

2.5) Let *TN* be the trigger name of any trigger *T* that is dependent on *A*. The following <drop trigger statement> is effectively executed for every *T*:

```
DROP TRIGGER TN
```

Insert the following Conformance Rule:

2) Conforming SQL language shall not contain an <drop assertion statement> that contains a <drop behavior>

11.38 <trigger definition>

1. Rationale: Clarify that the subject table of a <trigger definition> cannot be a declared local temporary table.

Replace Syntax Rule 5) with:

5) *T* shall be a base table that is not a declared local temporary table.

2. Rationale: Provide missing Access Rule.

Insert the following Access Rule:

3) If an <old or new values alias> has been specified, then the applicable privileges for *A* shall include SELECT on *T*.

3. Rationale: Editorial.

Replace Conformance Rule 1) with:

1) Without Feature T211, "Basic trigger capability", conforming SQL language shall not contain a <trigger definition>.

11.40 <user-defined type definition>

1. Rationale: Correct the specification of implicit generation of cast functions when reference values are generated via <user-defined representation>.

In the Format, replace the production of <user-defined type body> with:

```
<user-defined type body> ::=
     <user-defined type name>
     [ <subtype clause> ]
     [ AS <representation> ]
     [ <instantiable clause> ]
     <finality>
     [ <reference type specification> ]
     [ <ref cast option> ]
     [ <cast option> ]
     [ <method specification list> ]
```

In the Format, replace the production of <user-defined representation> with:

```
<user-defined representation> ::= REF USING <predefined type>
```

2. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values. Correct use of <specific name> for method specifications by replacing with <specific method name>.*

In the Format, replace the production for <partial method specification> with:

```
<partial method specification> ::=
     [ INSTANCE | STATIC | CONSTRUCTOR ] METHOD <method name>
     <SQL parameter declaration list>
     <returns clause>
     [ SPECIFIC <specific method name> ]
```

3. *Rationale: <method characteristic> should not include <transform group specification>.*

In the Format, replace the production for <method characteristic> with:

```
<method characteristic> ::=
       <language clause>
     | <parameter style clause>
     | <deterministic characteristic>
     | <SQL-data access indication>
     | <null-call clause>
```

4. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

In the Format, insert the following production:

```
<specific method name> ::=
     [ <schema name> <period> ] <qualified identifier>
```

5. *Rationale: Eliminate nonsensical combination of options.*

Insert the following Syntax Rule:

4.1) If <finality> specifies FINAL then <instantiable clause> shall not specify NOT INSTANTIABLE.

6. *Rationale: Change the second function definition to define an SQL-invoked regular function, not a method.*

Replace Syntax Rule 5) g) with:

5) g) If <cast to source> is specified, then let *FNSDT* be <cast to source identifier>; otherwise, the Syntax Rules of Subclause 9.7, ''Type name determination'', are applied to *SDT*, yielding an <identifier> *FNSDT*.

7. *Rationale: Prohibit a type from being a proper supertype of itself.*

Insert the following Syntax Rule:

6) i) i.0) <supertype name> shall not be equivalent to *UDTN*.

8. *Rationale: Remove unneeded rule during process of correcting the oversight in recursive type definition. Move check to <attribute definition>.*

Delete Syntax Rule 6) g) and associated NOTE 225.

9. *Rationale: Correct the specification of implicit generation of cast functions when reference values are generated via <user-defined representation>.*

Replace Syntax Rule 6) i) i) with:

6) i) i) Let *BT* be <predefined type>. *BT* is the *representation type of the referencing type* of *UDT*.

Insert the following Syntax Rule:

6) k) iii.1) If the user-defined type descriptor of *SST* indicates that the referencing type of *SST* has a user-defined representation, then let *BT* be the data type described by the data type descriptor of the representation type of the referencing type of *SST* included in the user-defined type descriptor of *SST*.

1) If <cast to ref> is specified, then let *FNREF* be <cast to ref identifier>; otherwise, let *FNREF* be the <qualified identifier> of *UDTN*.

2) Case:

A) If <cast to type> is specified, then let *FNTYP* be <cast to type identifier>.

B) Otherwise, the Syntax Rules of Subclause 9.7, "Type name determination", are applied to *BT*, yielding an <identifier> *FNTYP*.

Insert the following Syntax Rule:

6) k.1) If <ref cast option> is specified, then exactly one of the following shall be true:

i) <user-defined representation> is specified.

ii) <subtype clause> is specified and the user-defined type descriptor of the direct supertype of *UDT* indicates that the referencing type of the direct supertype of *UDT* has a user-defined representation.

10. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 7) a) with:

7) a) Let *M* be the number of <method specification>s MS_i, 1 (one) ≤ i ≤ M, contained in <method specification list>. Let MN_i be the <method name> of MS_i.

11. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Delete Syntax Rule 7) b) i)

12. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Delete Syntax Rule 7) b) ii)

13. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Replace Syntax Rule 7) b) iii) with:

7) b) iii) If MS_i does not specify INSTANCE, CONSTRUCTOR or STATIC, then INSTANCE is implicit.

14. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Insert the following Syntax Rule:

7) b) iv.1) If MS_i specifies CONSTRUCTOR, then:

1) SELF AS RESULT shall be specified.

2) OVERRIDING shall not be specified.

3) MN_i shall be equivalent to the <qualified identifier> of *UDTN*.

4) The <returns data type> shall specify *UDTN*.

4.1) *UDTD* shall define a structured type.

5) MS_i specifies an *SQL-invoked constructor method*.

15. Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.

Replace Syntax Rules 7) b) vi), 7) b) vii), 7) b) viii) with:

7) b) vi) If <specific method name> is not specified, then an implementation-dependent <specific method name> whose <schema name> is equivalent to *SN* is implicit.

7) b) vii) If <specific method name> contains a <schema name>, then that <schema name> shall be equivalent to *SN*. If <specific method name> does not contain a <schema name>, then the <schema name> of *SN* is implicit

7) b) viii) The schema identified by the explicit or implicit <schema name> of the <specific method name> shall not include a routine descriptor whose specific name is equivalent to <specific method name> or a user-defined type descriptor that includes a method specification descriptor whose specific method name is equivalent to <specific method name>.

16. Rationale: Remove the anomaly in <SQL parameter name>s.

Replace Syntax Rule 7) b) ix) with:

7) b) ix) Let PDL_i be the <SQL parameter declaration list> contained in MS_i.

1) No two <SQL parameter name>s contained in PDL_i shall be equivalent.

2) No <SQL parameter name> contained in PDL_i shall be equivalent to SELF.

17. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Insert the following Syntax Rule:

7) b) xiii) 1.1) Both MS_i and MS_k either specify CONSTRUCTOR, or both do not specify CONSTRUCTOR.

18. Rationale: <method characteristic> should not include <transform group specification>.

Replace Syntax Rule 7) b) xv) 1) with:

7) b) xv) 1) The <method characteristics> of MS_i shall contain at most one <language clause>, at most one <parameter style clause>, at most one <deterministic characteristic>, at most one <SQL-data access indication>, and at most one <null-call clause>.

19. Rationale: Clarify the semantics of SQL-data access indication.

Replace Syntax Rule 7) b) xv) 4) with:

7) b) xv) 4) <SQL-data access indication> shall be specified.

20. Rationale: <method characteristic> should not include <transform group specification>.

Replace Syntax Rule 7) b) xv) 6) A) III) with:

7) b) xv) 6) A) III) <parameter style clause> shall not be specified.

21. Rationale: <method characteristic> should not include <transform group specification>.

Delete Syntax Rule 7) b) xv) 6) B) II)

22. Rationale: The determination of conflicting methods should be based on data type compatibility rather than on data type identity.

Replace Syntax Rule 7) b) xv) 7) D) with:

7) b) xv) 7) D) For j varying from 1 (one) to N_i, the declared type of $PCMS_j$ and the declared type of PMS_{ij} are compatible.

23. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Insert the following Syntax Rule:

7) b) xv) 7) D.1) MS_i does not specify CONSTRUCTOR.

24. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 7) b) xv) 10):

7) b) xv) 10) If MS_i does not specify STATIC or CONSTRUCTOR, then there shall be no SQL-invoked function F that satisfies all the following conditions:

A) The routine name of F and RN_i have equivalent <qualified identifier>s.

B) If F is not a static method, then F has AN_i SQL parameters; otherwise, F has (AN_i - 1) SQL parameters.

C) The data type being defined is a proper subtype of

Case:

I) If F is not a static method, then the declared type of the first SQL parameter of F.

II) Otherwise, the user-defined type whose user-defined type descriptor includes the routine descriptor of F.

D) The declared type of the i-th SQL parameter in $NPLi$,$2 \le i \le AN_i$ is compatible with

Case:

I) If F is not a static method, then the declared type of i-th SQL parameter of F.

II) Otherwise, the declared type of the (i-1)-th SQL parameter of F.

25. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 7) b) xvi) 1) with:

7) b) xvi) 1) MS_i shall not specify STATIC or CONSTRUCTOR.

26. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Insert the following Syntax Rule:

7) b) xvi) 3) D.1) The descriptor of *COMS* shall not include an indication that STATIC or CONSTRUCTOR has been specified.

27. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 7) b) xvi) 5):

7) b) xvi) 5) *COMS* shall not be the corresponding method specification of a mutator or observer function.

28. *Rationale: For overriding method specifications, provide missing syntax rules to enforce proper <locator indication> in <SQL parameter declaration list>s. Also provide missing syntax rules to enforce that the parameter names are properly aligned with the parameter names of the corresponding parameters of the corresponding original method specification.*

Replace Syntax Rule 7) b) xvi) 6) with:

7) b) xvi) 6) For *j* ranging from 1 (one) to N_i, all of the following shall be true:

A) If the $POVMS_j$ contains an <SQL parameter name> *PNM1*, then $PCOMS_j$ contains an <SQL parameter name> that is equivalent to *PNM1*.

B) If $PCOMS_j$ contains an <SQL parameter name> *PNM2*, then $POVMS_j$ contains an <SQL parameter name> that is equivalent to *PNM2*.

C) If the $POVMS_j$ contains a <locator indication>, then $PCOMS_j$ contains a <locator indication>.

D) If $PCOMS_j$ contains a <locator indication>, then $POVMS_j$ contains a <locator indication>.

29. *Rationale: Supply the missing rules for checking the <returns data type> of overriding methods.*

Replace Syntax Rule 7) b) xvi) 7) with:

7) b) xvi) 7) Let *ROVMS* be the <returns data type> of MS_i. Let *RCOMS* be the <returns data type> of *COMS*.

Case:

A) If *RCOMS* is a user-defined type, then:

I) Let a candidate overriding method specification *COVRMS* be a method specification that is included in the descriptor of a proper supertype of *UDT*, such that the following are all true:

1) The <method name> of *COVRMS* and MN_i are equivalent.

2) *COVRMS* and MS_i have the same number of SQL-parameters N_i.

3) Let $PCOVRMS_i$, 1 (one) <= i <= Ni, be the i-th SQL parameter in the unaugmented SQL parameter declaration list of *COVRMS*. Let $POVMS_i$, 1 (one) <= i <= N_i, be the i-th SQL parameter in the unaugmented SQL parameter declaration list of *MS*i.

4) For i varying from 1 (one) to N_i, the Syntax Rules of Subclause 10.16, ''Data type identity'', are applied with the declared type of $PCOVRMS_i$ and the declared type of $POVMS_i$.

II) Let *NOVMS* be the number of candidate overriding method specifications. For i varying from 1 (one) to *NOVMS*, *ROVMS* shall be a subtype of the <returns data type> of i-th candidate overriding method specification.

B) Otherwise, the Syntax Rules of Subclause 10.16, ''Data type identity'', are applied with *RCOMS* and *ROVMS*.

30. Rationale: Do not allow distinct types to specify <ref cast option>.

Insert the following Syntax Rule:

7) e.1) <ref cast option> shall not be specified.

31. Rationale: Correct the first <user-defined cast definition> and the <transform definition>. Change method reference to reference an SQL-invoked regular function.

Replace General Rule 1) b) with:

1) b) The following SQL-statements are executed without further Access Rule checking:

```
CREATE FUNCTION SN.FNUDT ( SDTP SDT )
RETURNS UDTN
LANGUAGE SQL
DETERMINISTIC
RETURN RV1

CREATE FUNCTION SN.FNSDT ( UDTP UDTN )
RETURNS SDT
LANGUAGE SQL
DETERMINISTIC
RETURN RV2

CREATE CAST ( UDTN AS SDT )
WITH FUNCTION SN.FNSDT ( UDTN )
AS ASSIGNMENT

CREATE CAST ( SDT AS UDTN )
WITH FUNCTION SN.FNUDT ( SDT )
AS ASSIGNMENT

CREATE TRANSFORM FOR UDTN
FNUDT ( FROM SQL WITH FUNCTION SN.FNSDT ( UDTN ),
TO SQL WITH FUNCTION SN.FNUDT ( SDT ) )
```

where: *SN* is the explicit or implicit <schema name> of *UDTN*; *RV1* is an implementation-dependent <value expression> such that for every invocation of *SN.FNUDT* with argument value

AV1, *RV1* evaluates to the representation of *AV1* in the data type identified by *UDTN*; *RV2* is an implementation-dependent <value expression> such that for every invocation of *SN.FNSDT* with argument value *AV2*, *RV2* evaluates to the representation of *AV2* in the data type *SDT*, and *SDTP* and *UDTP* are <SQL parameter name> arbitrarily chosen.

32. *Rationale: Provide appropriate declarations for the system-generated ordering for distinct type whose source type is LOB. Change the second function definition to define an SQL-invoked regular function, not a method.*

Insert the following General Rule:

1) c) Case:

i) If *SDT* is not a large object type, then the following SQL-statement is executed without further Access Rule checking:

```
CREATE ORDERING FOR UDTN
ORDER FULL BY
MAP WITH FUNCTION FNSDT ( UDTN )
```

ii) If *SDT* is a large object type, and the SQL-implementation supports Feature T042, "Extended LOB data type support", then the following SQL-statement is executed without further Access Rule checking:

```
CREATE ORDERING FOR UDTN
ORDER EQUALS ONLY BY
MAP WITH FUNCTION FNSDT ( UDTN )
```

NOTE 228 — If *SDT* is a large object type, and the SQL-implementation does not support Feature T042, "Extended LOB data type support", then no ordering for *UDTN* is created.

33. *Rationale: Editorial.*

Replace General Rule 2) b) with:

2) b) If INSTANTIABLE is specified, then let *V* be a value of the most specific type *UDT* such that, for every attribute *ATT* of *UDT*, invocation of the corresponding observer function on *V* yields the default value for *ATT*. The following <SQL-invoked routine> is effectively executed:

```
CREATE FUNCTION UDTN ( ) RETURNS UDTN
RETURN V
```

This SQL-invoked function is the *constructor function* for *UDT*.

34. *Rationale: Correct the <SQL-invoked routine> statement that specifies SN.FNTYP and correct the last <user-defined cast definition>. Correct the specification of implicit generation of cast functions when reference values are generated via <user-defined representation>.*

Replace General Rule 2) c) with:

2) c) If <user-defined representation> is specified or if <subtype clause> is specified and the user-defined type descriptor of the direct supertype of *UDT* indicates that the referencing type of the

direct supertype of *UDT* has a user-defined representation, then the following SQL-statements are executed without further Access Rule checking:

```
CREATE FUNCTION SN.FNREF ( BTP BT )
  RETURNS REF ( UDTN )
  LANGUAGE SQL
  DETERMINISTIC
  STATIC DISPATCH
  RETURN RV1

CREATE FUNCTION SN.FNTYP ( UDTNP REF( UDTN ))
  RETURNS BT
  LANGUAGE SQL
  DETERMINISTIC
  STATIC DISPATCH
  RETURN RV2

CREATE CAST ( BT AS REF ( UDTN ) )
  WITH FUNCTION SN.FNREF ( BT )

CREATE CAST ( REF( UDTN ) AS BT )
  WITH FUNCTION SN.FNTYP ( REF( UDTN ) )
```

where: *SN* is the explicit or implicit <schema name> of *UDTN*; *RV1* is an implementation-dependent <value expression> such that for every invocation of *SN.FNREF* with argument value *AV1*, *RV1* evaluates to the representation of *AV1* in the data type identified by REF (*UDTN*); *RV2* is an implementation-dependent <value expression> such that for every invocation of *SN.FNTYP* with argument value *AV2*, *RV2* evaluates to the representation of *AV2* in the data type *BT*; and *UDTNP* is an <SQL parameter name> arbitrarily chosen.

35. *Rationale: Redundant subrule.*

Replace General Rule 5) e) with:

5) e) If *UDT* is a distinct type, then the data type descriptor of *SDT*.

36. *Rationale: Correct the specification of implicit generation of cast functions when reference values are generated via <user-defined representation>.*

Replace General Rule 5) f) v) with:

5) f) v) Case:

1) If <user-defined representation> is specified, then an indication that the referencing type of *UDT* has a user-defined representation and the data type descriptor of the representation type of the referencing type of *UDT*.

2) If <derived representation> is specified, then an indication that the referencing type of *UDT* has a derived representation, and the attributes specified by <list of attributes>.

3) Otherwise, an indication that the referencing type of *UDT* has a system-defined representation.

37. Rationale: Ordering form and category for a structured type.

Insert the following General Rules:

5) f) v.1) The ordering form NONE.

v.2) The ordering category STATE.

38. Rationale: Correct the specification of implicit generation of cast functions when reference values are generated via <user-defined representation>.

Insert the following General Rule:

5) f) v.3) If <subtype clause> is specified, then let *SUDT* be the direct supertype of *UDT* and let *DSUDT* be the user-defined type descriptor of *SUDT*. Let *RUDT* be the referencing type of *UDT* and let *RSUDT* be the referencing type of *SUDT*.

Case:

1) If *DSUDT* indicates that *RSUDT* has a user-defined representation, then an indication that *RUDT* has a user-defined representation and the data type descriptor of the representation type of *RSUDT* included in *DSUDT*.

2) If *DSUDT* indicates that *RSUDT* has a derived representation, then an indication that *RUDT* has a derived representation and the list of attributes included in *DSUDT*.

3) If *DSUDT* indicates that *RSUDT* has a system-defined representation, then an indication that *RUDT* has a system-defined representation.

39. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Replace General Rule 5) g) ii) with:

5) g) ii) An indication of whether STATIC or CONSTRUCTOR is specified.

40. Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.

Replace General Rule 5) g) iv) with:

5) g) iv) The <specific method name> of *ORMS*.

41. Rationale: <method characteristic> should not include <transform group specification>.

Delete General Rule 5) g) x)

42. Rationale: Clarify the semantics of SQL-data access indication.

Replace General Rule 5) g) xii) with:

5) g) xii) An indication of whether the method possibly modifies SQL data, possibly reads SQL-data, possibly contains SQL, or does not possibly contain SQL.

43. Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.

Replace General Rule 5) h) iii) with:

5) g) iii) The <specific method name> of *OVMS*.

44. Rationale: Supply the missing rules for checking the <returns data type> of overriding methods.

Replace General Rule 5) h) iv) with:

5) h) iv) The <SQL parameter declaration list> contained in *OVMS* (augmented to include the implicit first parameter with parameter name SELF).

45. Rationale: Supply the missing rules for checking the <returns data type> of overriding methods.

Replace General Rule 5) h) vii) with:

5) h) vii) The <returns data type> of *OVMS*.

46. Rationale: <method characteristic> should not include <transform group specification>.

Delete General Rule 5) h) ix)

47. Rationale: Clarify that Feature S023 also comprises <method specification list>.

Insert the following Conformance Rule:

8.1) Without Feature S023, "Basic structured types", conforming SQL language shall not specify <method specification list>.

11.41 <attribute definition>

1. Rationale: Correct oversight in rules for recursive type definition.

Insert the following Syntax Rule:

6.1) *DT* shall not be based on *UDT*.

NOTE 229.1 — The notion of one data type being based on another data type is defined in Subclause 4.1, "Data types".

2. Rationale: Raise more appropriate exception.

Replace General Rule 7) with:

7) An SQL-invoked method *MF* is created whose signature and result data type are as given in the descriptor of the original method specification of the mutator function of *A*. Let *V* be a value in *UDT*

and let *AV* be a value in *DT*. If *V* is the null value, then the invocation *V.AN(AV)* of *MF* raises an exception condition: *data exception — null value substituted for mutator subject parameter*; otherwise, the invocation *V.AN*(*AV*) returns *V2* such that *V2.AN*() = *AV* and for every other observer function *ANX* of *UDT*, *V2.ANX*() =*V.ANX*().

11.43 <add attribute definition>

1. Rationale: Correct oversight in rules for recursive type definition.

Replace Syntax Rule 3) with:

3) The declared type of a column of a base table shall not be SPRD, SBRD, SPAD, or SBAD.

2. Rationale: Correct oversight in rules for recursive type definition.

Replace Syntax Rule 4) with:

4) The declared type of a column of a base table shall not be based on D.

11.44 <drop attribute definition>

1. Rationale: Editorial.

Replace Syntax Rule 4) with:

4) Let *SPD* be any supertype of *D*. Let *SBD* be any subtype of *D*. Let *RD* be the reference type whose referenced type is *D*. Let *SPRD* be any supertype of *RD*. Let *SBRD* be any subtype of *RD*. Let *AD* be the array type whose element type is *D*. Let *SPAD* be any array type whose element type is *SPD* or *SPRD*. Let *SBAD* be any array type whose element type is *SBD* or *SBRD*.

2. Rationale: Correct oversight in rules for recursive type definition.

Replace Syntax Rule 5) with:

5) The declared type of any column of any base table shall not be *SPRD*, *SBRD*, *SPAD*, or *SBAD*.

3. Rationale: Correct oversight in rules for recursive type definition.

Replace Syntax Rule 6) with:

6) The declared type of any column of any base table shall not be based on *D*.

4. Rationale: Remove redundant and confusing references to assertion descriptor.

Replace Syntax Rule 8) a) iii) as follows:

8) a) iii) The <search condition> of any constraint descriptor.

5. *Rationale: Remove redundant and confusing references to assertion descriptor.*

Replace Syntax Rule 8) d) iii) as follows:

8) d) iii) The <search condition> of any constraint descriptor.

6. *Rationale: Dependency on a user-defined ordering requires only the existence of the ordering..*

Replace Syntax Rule 8) c) with:

8) c) *R1* or *R2* shall not be the ordering function in the descriptor of any user-defined type..

11.45 <add original method specification>

1. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Delete Syntax Rule 4)

2. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Delete Syntax Rule 5)

3. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 6) with:

6) If *PORMS* does not specify INSTANCE, CONSTRUCTOR or STATIC, then INSTANCE is implicit.

6.1) If *PORMS* specifies CONSTRUCTOR, then:

a) SELF AS RESULT shall be specified.

b) *MN* shall be equivalent to the <qualified identifier> of *DN*.

c) The <returns data type> shall specify *DN*.

c.1) *D* shall be a structured type.

d) *PORMS* specifies an *SQL-invoked constructor method*.

4. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace Syntax Rule 9) with:

9) Case:

a) If *PORMS* does not specify <specific method name>, then an implementation-dependent <specific method name> is implicit whose <schema name> is equivalent to *SN*.

b) Otherwise:

Case:

i) If <specific method name> contains a <schema name>, then that <schema name> shall be equivalent to *SN*.

ii) Otherwise, the <schema name> *SN* is implicit.

The schema identified by the explicit or implicit <schema name> of the <specific method name> shall not include a routine descriptor whose specific name is equivalent to <specific method name> or a user-defined type descriptor that includes a method specification descriptor whose specific method name is equivalent to <specific method name>.

5. *Rationale: <method characteristic> should not include <transform group specification>.*

Replace Syntax Rule 10) with:

10) *MCH* shall contain at most one <language clause>, at most one <parameter style clause>, at most one <deterministic characteristic>, at most one <SQL-data access indication>, and at most one <null-call clause>.

a) If <language clause> is not specified in *MCH*, then LANGUAGE SQL is implicit.

b) Case:

i) If LANGUAGE SQL is specified or implied, then:

1) <parameter style clause> shall not be specified.

2) <SQL-data access indication> shall not specify NO SQL.

3) Every <SQL parameter declaration> contained in <SQL parameter declaration list> shall contain an <SQL parameter name>.

4) The <returns clause> shall not specify a <result cast>.

ii) Otherwise:

1) If <parameter style clause> is not specified, then PARAMETER STYLE SQL is implicit.

2) If a <result cast> is specified, then let *V* be some value of the <data type> specified in the <result cast> and let *RT* be the <returns data type>. The following shall be valid according to the Syntax Rules of Subclause 6.22, "<cast specification>":

CAST (*V* AS *RT*)

3) If <result cast from type> *RCT* simply contains <locator indication>, then *RCT* shall be either binary large object type, character large object type, array type, or user-defined type.

c) If <deterministic characteristic> is not specified in *MCH*, then NOT DETERMINISTIC is implicit.

d) If <SQL-data access indication> is not specified, then CONTAINS SQL is implicit.

e) If <null call clause> is not specified in *MCH*, then CALLED ON NULL INPUT is implicit.

6. *Rationale: Remove the anomaly in <SQL parameter name>s.*

Replace Syntax Rule 11) with:

11) No two <SQL parameter name>s contained in *MPDL* shall be equivalent.

11.1) No <SQL parameter name> contained in *MPDL* shall be equivalent to SELF.

7. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values and the determination of conflicting methods should be based on data type compatibility rather than on data type identity.*

Replace Syntax Rule 15) with:

15) Case:

a) If *ORMS* specifies CONSTRUCTOR, then let a *conflicting method specification CMS* be a method specification whose descriptor is included in the descriptor of *D*, such that the following are all true:

i) *MPDL* and the unaugmented SQL parameter list of *CMS* have the same number *N* of SQL parameters.

ii) Let $PCMS_j$, 1 (one) ≤ *j* ≤ *N*, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list of *CMS*. Let PMS_j, 1 (one) ≤ *j* ≤ *N*, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list *MPDL*.

iii) For *j* varying from 1 (one) to *N* the declared type of $PCMS_j$ and the declared type of PMS_j are compatible.

iv) *CMS* is an SQL-invoked constructor method.

b) Otherwise, let a *conflicting method specification CMS* be a method specification whose descriptor is included in the descriptor of some *SPD* or *SBD*, such that the following are all true:

i) *MN* and the method name included in the descriptor of *CMS* are equivalent.

ii) *MPDL* and the unaugmented SQL parameter list of *CMS* have the same number *N* of SQL parameters.

iii) Let $PCMS_j$, 1 (one) ≤ j ≤ *N*, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list of *CMS*. Let PMS_j, 1 (one) ≤ *j* ≤ *N*, be the *j*-th SQL parameter in the unaugmented SQL parameter declaration list *MPDL*.

iv) For *j* varying from 1 (one) to *N* the declared type of $PCMS_j$ and the declared type of PMS_j are compatible.

v) *CMS* and *ORMS* either both are instance methods or one of *CMS* and *ORMS* is a static method and the other is an instance method.

8. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 18) with:

18) If *PORMS* does not specify STATIC or CONSTRUCTOR, then there shall be no SQL-invoked function *F* that satisfies all the following conditions:

a) *F* is not an SQL-invoked method.

b) The <routine name> of *F* and *RN* have equivalent <qualified identifier>s.

c) *F* has *AN* SQL parameters.

d) *D* is a subtype or supertype of the declared type of the first SQL parameter of *F*.

e) The declared type of the *i*-th SQL parameter in *NPL*, $2 \leq i \leq AN$ is compatible with the declared type of *i*-th SQL parameter of *F*.

9. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace General Rule 1 b) with:

1) b) An indication of whether STATIC or CONSTRUCTOR is specified.

10. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace General Rule 1) d) with:

1) d) The <specific method name> of *PORMS*.

11. *Rationale: <method characteristic> should not include <transform group specification>.*

Delete General Rule 1) j)

12. *Rationale: Clarify the semantics of SQL-data access indication.*

Replace General Rule 1) n) with:

1) n) An indication of whether the method possibly modifies SQL data, possibly reads SQL-data, possibly contains SQL, or does not possibly contain SQL.

13. *Rationale: Create table/method privileges for tables defined on the subject type of the method being added.*

Insert the following General Rule:

2.1) Let *N* be the number of table descriptors that include the user-defined type name of a subtype of *D*.

For *i* varying from 1 (one) to *N*,

a) Let TN_i be the <table name> included in the *i*-th such table descriptor.

b) For every table privilege descriptor that specifies TN_i and a privilege of SELECT, a new table/method privilege descriptor is created that specifies TN_i, the same action, grantor, and grantee, and the same grantability, and the <specific method name> contained in *ORMS*.

11.46 <add overriding method specification>

1. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 2) with:

2) Let *POVMS* be the <partial method specification> immediately contained in *OVMS*. *POVMS* shall not specify STATIC or CONSTRUCTOR.

2. *Rationale: Remove syntax rule that conflicts with format rules.*

Delete Syntax Rule 5)

3. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace Syntax Rule 7) with:

7) Case:

a) If *POVMS* does not specify <specific method name>, then an implementation-dependent <specific method name> is implicit whose <schema name> is equivalent to *SN*.

b) Otherwise:

Case:

i) If <specific method name> contains a <schema name>, then that <schema name> shall be equivalent to *SN*.

ii) Otherwise, the <schema name> *SN* is implicit.

The schema identified by the explicit or implicit <schema name> of the <specific method name> shall not include a routine descriptor whose specific name is equivalent to <specific method name> or a user-defined type descriptor that includes a method specification descriptor whose specific method name is equivalent to <specific method name>.

4. *Rationale: Correct improper use of terms in Syntax Rule 9) a).*

Replace Syntax Rule 9) a) with:

9) a) *MN* and the <method name> of *COMS* are equivalent.

5. *Rationale: Remove improper use of "shall".*

Replace Syntax Rule 9) b) with:

9) b) Let *N* be the number of elements of the augmented SQL parameter declaration list *UPCOMS* generally included in the descriptor of *COMS*. *MPDL* contains (*N*-1) SQL parameter declarations.

6. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Insert the following Syntax Rule:

9) c.1) The descriptor of *COMS* shall not include an indication that STATIC or CONSTRUCTOR has been specified.

7. *Rationale: Rectify misplacement of Syntax Rule 9) d); correct the subrules i) and iv) to enforce correct match of parameter names and locator indications.*

Delete Syntax Rule 9) d)

8. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Delete Syntax Rule 11)

9. *Rationale: Rectify misplacement of Syntax Rule 9) d); correct the subrules i) and iv) to enforce correct match of parameter names and locator indications.*

Insert the following Syntax Rule:

12.1) For *i* varying from 2 to *N*:

a) If $POVMS_{i-1}$ contains an <SQL parameter name> *PNM1*, then the descriptor of the *i*-th parameter of the augmented <SQL parameter declaration list> of *UPCOMS* shall include a parameter name that is equivalent to *PNM1*.

b) If the descriptor of the *i*-th parameter of the augmented <SQL parameter declaration list> of *UPCOMS* includes a parameter name *PNM2*, then $POVMS_{i-1}$ shall contain an <SQL parameter name> that is equivalent to *PNM2*.

c) $POVMS_{i-1}$ shall not contain <parameter mode>. A <parameter mode> IN is implicit.

d) $POVMS_{i-1}$ shall not specify RESULT.

e) If the <parameter type> PT_{i-1} immediately contained in $POVMS_{i-1}$ contains a <locator indication>, then the descriptor of the *i*-th parameter of the augmented <SQL parameter declaration list> of *UPCOMS* shall include a <locator indication>.

f) If the descriptor of the *i*-th parameter of the augmented <SQL parameter declaration list> of *UPCOMS* includes a <locator indication>, then the <parameter type> PT_{i-1} immediately contained in $POVMS_{i-1}$ shall contain a <locator indication>.

10. Rationale: Supply the missing rules for checking the <returns data type> of overriding methods.

Replace Syntax Rule 13) with:

13) Let *ROVMS* be the <returns data type> of *RTC*. Let *RCOMS* be the <returns data type> of *COMS*.

Case:

a) If *RCOMS* is a user-defined type, then:

i) Let a candidate overriding method specification *COVRMS* be a method specification that is included in the descriptor of a proper supertype or a proper subtype of *D*, such that the following are all true:

1) The <method name> of *COVRMS* and *MN* are equivalent.

2) *COVRMS* and *OVMS* have the same number of SQL-parameters N_i.

3) Let $PCOVRMS_i$, 1 (one) <= i <= N_i, be the i-th SQL parameter in the unaugmented SQL parameter declaration list of *COVRMS*. Let $POVMS_i$, 1 (one) <= i <= N_i, be the i-th SQL parameter in the unaugmented SQL parameter declaration list of *OVMS*.

4) For i varying from 1 (one) to N_i, the Syntax Rules of Subclause 10.16, ''Data type identity'', are applied with the declared type of $PCOVRMS_i$ and the declared type of $POVMS_i$.

ii) Let *NOVMS* be the number of candidate overriding method specifications. For i varying from 1 (one) to *NOVMS*, let $COVRMS_i$ be the i-th candidate overriding method specification.

Case:

1) If $COVRMS_i$ is included in the descriptor of a proper supertype of *D*, then *ROVMS* shall be a subtype of the <returns data type> of $COVRMS_i$.

2) Otherwise, *ROVMS* shall be a supertype of the <returns data type> of $COVRMS_i$.

b) Otherwise, the Syntax Rules of Subclause 10.16, ''Data type identity'', are applied with RCOMS and *ROVMS* as the data types.

11. Rationale: Add missing syntax rule that prevents the addition of a conflicting overriding method specification to some structured type.

Insert the following Syntax Rule:

13.1) Let a *conflicting overriding method specification COVMS* be an overriding method specification that is included in the descriptor of *D*, such that the following are all true:

a) *MN* and the method name of *COVMS* are equivalent.

b) The augmented SQL parameter declaration list of *COVMS* contains *N* elements.

c) For *i* varying from 2 to *N*, the data types of the SQL parameter *POVMSi-1* and the SQL parameter *PCOVMSi* of *COVMS* are compatible.

There shall be no conflicting overriding method specification *COVMS*.

12. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Insert the following Syntax Rule:

16) b) ii) 3.1) MSD_i does not include an indication that CONSTRUCTOR has been specified.

13. Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.

Replace General Rule 1) c) with:

1) c) The <specific method name> of *POVMS*.

14. Rationale: Supply the missing rules for checking the <returns data type> of overriding methods.

Replace General Rule 1) h) with:

1) h) The <returns data type> contained in *POVMS*.

15. Rationale: <method characteristic> should not include <transform group specification>.

Delete General Rule 1) j)

16. Rationale: Create table/method privileges for tables defined on the subject type of the method being added.

Insert the following General Rule:

2.1) Let *N* be the number of table descriptors that include the user-defined type name of a subtype of *D*.

For *i* varying from 1 (one) to *N*,

a) Let TN_i be the <table name> included in the *i*-th such table descriptor.

b) Let *M* be the number of table/method privilege descriptors that specify TN_i and the <specific method name> contained in *COMS*. For *j* varying from 1 (one) to *M*,

i) Let $TMPD_j$ be the *j*-th such table/method privilege descriptor.

ii) A new table/method privilege descriptor is created that specifies TN_i, the same action, grantor, and grantee, and the same grantability, and the <specific method name> contained in *OVMS*.

iii) $TMPD_j$ is deleted.

11.47 <drop method specification>

1. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace the Format with:

```
<drop method specification> ::=
     DROP <specific method specification designator> RESTRICT

<specific method specification designator> ::=
       SPECIFIC METHOD <specific method name>
     | [ INSTANCE | STATIC | CONSTRUCTOR] METHOD <method name>
         [ <data type list> ]
```

2. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace the Syntax Rules with:

1) Let *D* be the user-defined type identified by the <user-defined type name> *DN* immediately contained in the <alter type statement> containing the <drop method specification> *DORMS*. Let *DSN* be the explicit or implicit <schema name> of *DN*. Let *SMSD* be the <specific method specification designator> immediately contained in *DORMS*.

2) If *SMSD* immediately contains a <specific method name> *SMN*, then

   a) If <specific method name> contains a <schema name>, then that <schema name> shall be equivalent to *DSN*. If <specific method name> does not contain a <schema name>, then the <schema name> *DSN* is implicit.

   b) The descriptor of *D* shall include a method specification descriptor *DOOMS* whose specific method name is equivalent to *SMN*.

   c) Let *PDL* be the augmented parameter list included in *DOOMS*.

   d) Let *MN* be the <method name> included in *DOOMS*.

3) If *SMSD* immediately contains a <method name> *ME*, then:

   a) If none of INSTANCE, STATIC, or CONSTRUCTOR is immediately contained in *SMSD*, then INSTANCE is implicit.

   b) The descriptor of *D* shall include a method specification descriptor *DOOMS* whose method name *MN* is equivalent to *ME*.

   c) If *SMSD* immediately contains a <data type list> *DTL*, then

      Case:

      i) If STATIC is specified, then the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* which includes:

A) An indication that the method specification is STATIC.

B) An indication that the method specification is original.

C) An augmented parameter list *PDL* such that for all *i*, the declared type of its *i*-th parameter is identical to the *i*-th declared type in *DTL*.

ii) If CONSTRUCTOR is specified, then the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* which includes:

A) An indication that the method specification is CONSTRUCTOR.

B) An indication that the method specification is original.

C) An augmented parameter list *PDL* such that for all $i > 1$, the declared type of its *i*-th parameter is identical to the *(i -1)*-th declared type in *DTL* and the declared type of the first parameter of *PDL* is identical to *DN*.

iii) Otherwise, the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* for which:

A) If *DOOMS* includes an indication that the method specification is original, then *DOOMS* shall not include an indication that the method specification is either STATIC or CONSTRUCTOR.

B) *DOOMS* includes an augmented parameter list *PDL* such that for all $i > 1$, the declared type of its *i*-th parameter is identical to the *(i -1)*-th declared type in *DTL* and the declared type of the first parameter of *PDL* is identical to *DN*.

d) If *SMSD* does not immediately contain a <data type list>, then:

Case:

i) If STATIC is specified, the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* which includes an indications that the method specification is both original and STATIC.

ii) If CONSTRUCTOR is specified, the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* which includes an indications that the method specification is both original and CONSTRUCTOR.

iii) Otherwise, the descriptor of *D* shall include exactly one method specification descriptor *DOOMS* for which: if *DOOMS* includes an indication that the method specification is original, then *DOOMS* shall not include an indication that the method specification is either STATIC or CONSTRUCTOR.

4) Case:

a) If *DOOMS* includes an indication that the method specification is original, then:

Case:

i) If *DOOMS* includes an indication that the method specification specified STATIC, then there shall be no SQL-invoked function *F* that satisfies all of the following conditions:

1) The <routine name> of *F* and *MN* have equivalent <qualified identifier>s.

2) If *N* is the number of elements in *PDL*, then *F* has *N* SQL parameters.

3) The declared type of the first SQL parameter of *F* is *D*.

4) The declared type of the *i*-th element of *PDL*, 1 <= *i* <= *N*, is compatible with the declared type of SQL parameter P_i of *F*.

5) *F* is an SQL-invoked method.

6) *F* includes an indication that STATIC is specified.

ii) If *DOOMS* includes an indication that the method specification specified CONSTRUCTOR, then there shall be no SQL-invoked function *F* that satisfies all of the following conditions:

1) The <routine name> of *F* and *MN* have equivalent <qualified identifier>s.

2) If *N* is the number of elements in *PDL*, then *F* has *N* SQL parameters.

3) The declared type of the first SQL parameter of *F* is *D*.

4) The declared type of the *i*-th element of *PDL*, 2 <= *i* <= *N*, is compatible with the declared type of SQL parameter P_i of *F*.

5) *F* is an SQL-invoked method.

6) *F* includes an indication that CONSTRUCTOR is specified.

iii) Otherwise:

1) There shall be no proper subtype *PSBD* of *D* whose descriptor includes the descriptor *DOVMS* of an overriding method specification such that all of the following is true:

A) *MN* and the < method name> included in *DOVMS* have equivalent <qualified identifier>s.

B) If *N* is the number of elements in *PDL*, then the augmented SQL parameter declaration list *APDL* included in *DOVMS* has *N* SQL parameters.

C) *PSBD* is the declared type the first SQL parameter of *APDL*.

D) The declared type of the *i*-th element of *PDL*, 2 <= *i* <= *N*, is compatible with the declared type of SQL-parameter P_i of *APDL*.

2) There shall be no SQL-invoked function *F* that satisfies all of the following conditions:

A) The <routine name> of *F* and *MN* have equivalent <qualified identifier>s.

B) If *N* is the number of elements in *PDL*, then *F* has *N* SQL parameters.

C) The declared type of the first SQL parameter of *F* is *D*.

D) The declared type of the *i*-th element of *PDL*, 2 <= *i* <= *N*, is compatible with the declared type of SQL parameter P_i of *F*.

E) *F* is an SQL-invoked method.

F) *F* does not include an indication that either STATIC or CONSTRUCTOR is specified.

b) Otherwise, there shall be no SQL-invoked function *F* that satisfies all of the following conditions:

i) The <routine name> of *F* and *MN* have equivalent <qualified identifier>s.

ii) If *N* is the number of elements in *PDL*, then *F* has *N* SQL parameters.

iii) The declared type of the first SQL parameter of *F* is *D*.

iv) The declared type of the *i*-th element of *PDL*, 2 <= *i* <= *N*, is compatible with the declared type of SQL parameter P_i of *F*.

v) *F* is an SQL-invoked method.

vi) *F* does not include an indication that either STATIC or CONSTRUCTOR is specified.

Replace the General Rules with:

1) Let *STDS* be the descriptor of *D*.

2) *DOOMS* is removed from *STDS*.

3) *DOOMS* is destroyed.

11.48 <drop data type statement>

1. Rationale: Correct oversight in rules for recursive type definition.

Replace Syntax Rule 4) a) with:

4) a) The declared type of no column, field, or attribute whose descriptor is not included in the descriptor of *D* shall be based on *SRD*, or *SAD*.

2. Rationale: Correct oversight in rules for recursive type definition.

Replace Syntax Rule 4) b) with.

4) b) The declared type of no column, field, or attribute shall be based on *D*.

3. *Rationale: Remove redundant and confusing references to assertion descriptor.*

Replace Syntax Rule 4) f) ii) as follows:

4) f) ii) The <search condition> of any constraint descriptor.

4. *Rationale: Remove redundant and confusing references to assertion descriptor.*

Replace Syntax Rule 4) h) i) 3) as follows:

4) h) i) 3) The <search condition> of any constraint descriptor.

5. *Rationale: Dependency on a user-defined ordering requires only the existence of the ordering.*

Replace Syntax Rule 4) h) iii) with:

4) h) iii) *R* shall not be the ordering function included in the descriptor of any user-defined type.

Delete Syntax Rule 4) h) iv).

6. *Rationale: Correct the syntax of the executed <revoke statement>.*

Replace General Rule 7) with:

7) For every privilege descriptor that references *D*, the following <revoke statement> is effectively executed:

REVOKE *PRIV* ON TYPE *D* FROM *GRANTEE* CASCADE

where *PRIV* and *GRANTEE* are respectively the action and grantee in the privilege descriptor.

11.49 <SQL-invoked routine>

1. *Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears. Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

In the Format, replace the production for <method specification designator> with:

```
<method specification designator> ::=
    [ INSTANCE | STATIC | CONSTRUCTOR ] METHOD <method name>
    <SQL parameter declaration list>
    [ <returns clause> ]
    FOR <user-defined type name>
```

2. *Rationale: <routine characteristic> should not include <transform group specification>.*

In the Format, replace the production for <routine characteristic> with:

```
<routine characteristic> ::=
       <language clause>
     | <parameter style clause>
     | SPECIFIC <specific name>
     | <deterministic characteristic>
     | <SQL-data access indication>
     | <null-call clause>
     | <dynamic result sets characteristic>
```

3. *Rationale: <external body reference> should include <transform group specification>.*

In the Format, replace the production for <external body reference> with:

```
<external body reference> ::=
     EXTERNAL [ NAME <external routine name> ]
     [ <parameter style clause> ]
     [ <transform group specification> ]
     [ <external security clause> ]
```

4. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 3) with:

3) An <SQL-invoked routine> specified as an <SQL-invoked procedure> is called an *SQL-invoked procedure*; an <SQL-invoked routine> specified as an <SQL-invoked function> is called *an SQL-invoked function*. An <SQL-invoked function> that specifies a <method specification designator> is further called an *SQL-invoked method*. An SQL-invoked method that specifies STATIC is called a *static SQL-invoked method*. An SQL-invoked method that specifies CONSTRUCTOR is called an *SQL-invoked constructor method*.

5. *Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears.*

Replace Syntax Rule 4) a) with:

4) a) Let *UDTN* be the <user-defined type name> immediately contained in <method specification designator>. Let *UDT* be the user-defined type identified by *UDTN*.

6. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.*

Replace Syntax Rule 4) b) with:

4) b) There shall exist a method specification descriptor *DMS* in the descriptor of *UDT* such that the <method name> of *DMS* is equivalent to the <method name>, *DMS* indicates STATIC if and only if the <method specification designator> specifies STATIC, *DMS* indicates CONSTRUCTOR if and only if the <method specification designator> specifies CONSTRUCTOR and the declared type of every SQL parameter in the unaugmented SQL parameter declaration list in *DMS* is

compatible with the declared type of the corresponding SQL parameter in the <SQL parameter declaration list> contained in the <method specification designator>. *DMS* identifies the corresponding method specification of the <method specification designator>.

7. *Rationale: Remove the anomaly in <SQL parameter name>s.*

Replace Syntax Rule 4) e) with:

4) e) For *i* varying from 1 (one) to *MN*, the <SQL parameter name>s contained in $PCOMS_i$ and $POVMS_i$ shall be equivalent.

8. *Rationale: <method characteristic> should not include <transform group specification>.*

Delete Syntax Rule 4) j).

9. *Rationale: Correct use of <specific name> for method specifications by replacing with <specific method name>.*

Replace Syntax Rule 4) k) with:

4) k) Let *SPN* be the <specific method name> in *DMS*. *SPN* is the <specific name> of *R*.

10. *Rationale: <method characteristic> should not include <transform group specification>.*

Replace Syntax Rule 5) a) with:

5) a) <routine characteristics> shall contain at most one <language clause>, at most one <parameter style clause>, at most one <specific name>, at most one <deterministic characteristic>, at most one <SQL-data access indication>, at most one <null-call clause>, and at most one <dynamic result sets characteristic>.

11. *Rationale: Clarify the semantics of SQL-data access indication.*

Replace Syntax Rule 5) g) with:

5) g) <SQL-data access indication> shall be specified.

12. *Rationale: <method characteristic> should not include <transform group specification>.*

Replace Syntax Rule 5) j) iii) with:

5) ~~j)~~ iii) <parameter style clause> shall not be specified.

13. *Rationale: Remove the anomaly in <SQL parameter name>s.*

Replace Syntax Rule 10) with:

10) No two <SQL parameter name>s contained in *NPL* shall be equivalent.

14. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Insert the following Syntax Rule:

15) b) i) 1.1) If *R* is an SQL-invoked constructor method, then let *SCR* be the set containing every SQL-invoked constructor method of type *UDT*, including *R*, whose <schema qualified routine name> is equivalent to *RN* and whose number of SQL parameters is *PN*.

15. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Replace Syntax Rule 15) b) i) 2) with:

15) b) i) 2) Otherwise, let *SCR* be the set containing every SQL-invoked function in *S* that is not a static SQL-invoked method or an SQL-invoked constructor method, including *R*, whose <schema qualified routine name> is equivalent to *RN* and whose number of SQL parameters is *PN*.

16. Rationale: Clarify the semantics of SQL-data access indication.

Replace Syntax Rule 18) c) with:

18) c) If READS SQL DATA is specified, then it is implementation-defined whether the <SQL routine body> shall not contain an <SQL procedure statement> *S* that satisfies at least one of the following:

i) *S* is an <SQL data change statement>.

ii) *S* contains a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data.

iii) *S* contains an <SQL procedure statement> that is an <SQL data change statement>.

Replace Syntax Rule 18) d) with:

18) d) If CONTAINS SQL is specified, then it is implementation-defined whether the <SQL routine body> shall not contain an <SQL procedure statement> *S* that satisfies at least one of the following:

i) *S* is an <SQL data statement> other than <free locator statement> and <hold locator statement>.

ii) *S* contains a <routine invocation> whose subject routine is an SQL-invoked routine that possibly modifies SQL-data or possibly reads SQL-data.

iii) *S* contains an <SQL procedure statement> that is an <SQL data statement> other than <free locator statement> and <hold locator statement>.

17. *Rationale: Eliminate a syntactic ambiguity that can occur when a <schema definition> is contained in an <SQL routine body>.*

18) h) An <SQL routine body> shall not immediately contain an <SQL procedure statement> that simply contains a <schema definition>.

18. *Rationale: Clarify that the implicit schema of user-defined type used as a return type or parameter type is resolved using the SQL-path. Correct SR 19)d), which uses the Case construct improperly.*

Replace Syntax Rule 19)d) with:

19) d) If PARAMETER STYLE SQL is specified, then:

i) If *R* is an SQL-invoked function, then let *FRN* be 1 (one).

ii) If *R* is an array-returning external function, then let *AREF* be 7. Otherwise, let *AREF* be 5.

iii) If *R* is an SQL-invoked function, then let the *effective SQL parameter list* be a list of *PN+FRN+N+AREF* SQL parameters, as follows:

1) For *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is defined as follows.

Case:

A) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> contains <locator indication>, then the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by INTEGER.

B) If the <parameter type> T_i immediately contained in the *i*-th <SQL parameter declaration> is a <user-defined type> without a <locator indication>, then:

I) Case:

1) If *R* is an SQL-invoked method that is an overriding method, then the Syntax Rules of Subclause 10.16, ''Determination of a from-sql function for an overriding method'', are applied with *R* and *i* as *ROUTINE* and *POSITION*, respectively. There shall be an applicable from-sql function FSF_i.

2) Otherwise, the Syntax Rules of Subclause 10.15, ''Determination of a from-sql function'', are applied with the data type identified by T_i, and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable from-sql function FSF_i.

II) FSF_i is called the *from-sql function associated with the i-th SQL parameter*.

III) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by the <returns data type> of FSF_i.

C) Otherwise, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

2) Effective SQL parameter list entry *PN*+*FRN* has <parameter mode> OUT; its <parameter type> *PT* is defined as follows:

A) If <result cast> is specified, then let *RT* be <result cast from type>; otherwise, let *RT* be <returns data type>.

B) Case:

I) If *RT* simply contains <locator indication>, then *PT* is INTEGER.

II) If *RT* specifies a <user-defined type> without a <locator indication>

1) Case:

a) If *R* is an SQL-invoked method that is an overriding method, then the Syntax Rules of Subclause 10.18, "Determination of a to-sql function for an overriding method", are applied with *R* as *ROUTINE*.
There shall be an applicable to-sql function *TSF*.

b) Otherwise, the Syntax Rules of Subclause 10.17, "Determination of a to-sql function", are applied with the data type identified by *RT* and the <group name> contained in the <group specification> that contains *RT* as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function *TSF*.

2) *TSF* is called the to-sql function associated with the result.

3) Case:

a) If *TSF* is an SQL-invoked method, then *PT* is the <parameter type> of the second SQL parameter of *TSF*.

b) Otherwise, *PT* is the <parameter type> of the first SQL parameter of *TSF*.

III) If *R* is an array-returning external function, then let *PT* be the element type of *RT*.

IV) Otherwise, *PT* is *RT*.

3) Effective SQL parameter list entries (*PN*+*FRN*)+1 to (*PN*+*FRN*)+*N*+1 are *N*+1 occurrences of SQL parameters of an implementation-defined <data type> that is an exact numeric type with scale 0. For *i* ranging from (*PN*+*FRN*)+1 to (*PN*+*FRN*)+*N*+1, the <parameter mode> for the *i*-th such effective SQL parameter is the same as that of the *i*-*FRN*-*PN*-th effective SQL parameter.

4) Effective SQL parameter list entry (*PN*+*FRN*)+(*N*+1)+1 is an SQL parameter of a <data type> that is character string of length 5 and the character set specified for SQLSTATE values, with <parameter mode> INOUT.

NOTE 245 – The character set specified for SQLSTATE values is defined in Subclause 22.1, ''SQLSTATE''.

5) Effective SQL parameter list entry (*PN*+*FRN*)+(*N*+1)+2 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

6) Effective SQL parameter list entry (*PN*+*FRN*)+(*N*+1)+3 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

7) Effective SQL parameter list entry (*PN*+*FRN*)+(*N*+1)+4 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

8) If *R* is an array-returning external function, then:

A) Effective SQL parameter type list entry (*PN*+*FRN*)+(*N*+1)+5 is an SQL parameter whose <data type> is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

B) Effective SQL parameter type list entry (*PN*+*FRN*)+(*N*+1)+5 is an SQL parameter whose <data type> is an exact numeric type with scale 0 (zero) and with <parameter mode> IN.

iv) If *R* is an SQL-invoked procedure, then let the *effective SQL parameter list* be a list of *PN*+*N*+4 SQL parameters, as follows:

1) For *i* ranging from 1 (one) to *PN*, the *i*-th effective SQL parameter list entry is defined as follows.

Case:

A) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> simply contains <locator indication>, then the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by INTEGER.

B) If the <parameter type> T_i simply contained in the *i*-th <SQL parameter declaration> is a <user-defined type> without a <locator indication>, then:

I) Case:

1) If the <parameter mode> immediately contained in the *i*-th <SQL parameter declaration> is IN, then:

a) The Syntax Rules of Subclause 10.15, "Determination of a from-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable from-sql function FSF_i. FSF_i is called the from-sql function associated with the *i*-th SQL parameter.

b) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by the <returns data type> of FSF_i.

2) If the <parameter mode> immediately contained in the *i*-th <SQL parameter declaration> is OUT, then:

a) The Syntax Rules of Subclause 10.17, "Determination of a to-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function TSF_i. TSF_i is called the to-sql function associated with *i*-th SQL parameter.

b) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by

Case:

i) If TSF_i is an SQL-invoked method, then the <parameter type> of the second SQL parameter of TSF_i .

ii) Otherwise, the <parameter type> of the first SQL parameter of *TSFi* .

3) Otherwise:

a) The Syntax Rules of Subclause 10.15, "Determination of a from-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable from-sql function FSF_i . FSF_i is called the from-sql function associated with the *i*-th SQL parameter.

b) The Syntax Rules of Subclause 10.17, "Determination of a to-sql function", are applied with the data type identified by T_i and the <group name> contained in the <group specification> that contains T_i as *TYPE* and *GROUP*, respectively. There shall be an applicable to-sql function TSF_i. TSF_i is called the to-sql function associated with the *i*-th SQL parameter.

c) The *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration> with the <parameter type> replaced by the <returns data type> of FSF_i.

C) Otherwise, the *i*-th effective SQL parameter list entry is the *i*-th <SQL parameter declaration>.

2) Effective SQL parameter list entries *PN*+1 to *PN*+*N* are *N* occurrences of an SQL parameter of an implementation-defined <data type> that is an exact numeric type with scale 0. The <parameter mode> for the *i*-th such effective SQL parameter is the same as that of the *i*-*PN*-th effective SQL parameter.

3) Effective SQL parameter list entry (*PN*+*N*)+1 is an SQL parameter of a <data type> that is character string of length 5 and character set SQL_TEXT with <parameter mode> INOUT.

4) Effective SQL parameter list entry (*PN*+*N*)+2 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

5) Effective SQL parameter list entry (*PN*+*N*)+3 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> IN.

6) Effective SQL parameter list entry (*PN*+*N*)+4 is an SQL parameter of a <data type> that is character string of implementation-defined length and character set SQL_TEXT with <parameter mode> INOUT.

19. Rationale: Clarify the semantics of SQL-data access indication.

Replace Syntax Rule 20) with:

20) Case:

a) If <method specification designator> is specified, then:

i) If *DMS* indicates that the method is deterministic, then *R* is deterministic; otherwise, *R* is possibly non-deterministic.

ii) If the SQL-data access indication of *DMS* indicates that the method possibly modifies SQL-data, then *R* possibly modifies SQL-data. If the SQL-data access indication of *DMS* indicates that the method possibly reads SQL-data, then *R* possibly reads SQL-data. If the SQL-data access indication of *DMS* indicates that the method possibly contains SQL, then *R* possibly contains SQL. Otherwise, *R* does not possibly contain SQL.

b) Otherwise:

i) If DETERMINISTIC is specified, then *R* is deterministic; otherwise, it is possibly non-deterministic.

ii) An <SQL-invoked routine> possibly modifies SQL-data if and only if <SQL-data access indication> specifies MODIFIES SQL DATA.

iii) An <SQL-invoked routine> possibly reads SQL-data if and only if <SQL-data access indication> specifies READS SQL DATA.

iv) An <SQL-invoked routine> possibly contains SQL if and only if <SQL-data access indication> specifies CONTAINS SQL.

Replace General Rule 3) o) with:

3) o) The SQL-invoked routine descriptor includes an indication whether the SQL-invoked routine does not possibly contain SQL, possibly contains SQL, possibly reads SQL-data, or possibly modifies SQL-data.

20. Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values.

Replace General Rule 4) d) with:

4) d) The routine descriptor includes an indication that the SQL-invoked routine is an SQL-invoked function that is an SQL-invoked method, an indication of the user-defined type *UDT*, and an indication of whether STATIC or CONSTRUCTOR was specified.

21. Rationale: Editorial.

Replace Conformance Rules 1) and 4) with:

1) Without Feature T471, "Result sets return value", conforming SQL language shall not specify <dynamic result sets characteristic>.

4) Without Feature S241, "Transform functions", conforming SQL language shall not specify <transform group specification>.

22. Rationale: Remove useless item from Core.

Insert the following Conformance Rule:

10.1 Without at least one of the features:
S231 "Structured Type locators",
S232 "Array Locators",
T041 "Basic LOB data type"
conforming SQL language shall not specify <locator indication>.

11.50 <alter routine statement>

1. Rationale: Dependency on a user-defined ordering requires only the existence of the ordering..

Replace Syntax Rule 4) a) with:

4) a) *SR* shall not be the ordering function included in the descriptor of any user-defined type *UDT*.

2. Rationale: Remove redundant and confusing references to assertion descriptor.

Replace Syntax Rule 4) b) iii) as follows:

4) b) iii) The <search condition> of any constraint descriptor.

3. Rationale: Clarify that <language clause> shall not specify SQL.

Replace Syntax Rule 9) with:

9) If <language clause> is specified, then:

a) <language clause> shall not specify SQL.

b) Depending on whether the <language clause> specifies ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, let the operative data type correspondences table be Table 18, "Data type correspondences for Ada", Table 19, "Data type correspondences for C", Table 20, "Data type correspondences for COBOL", Table 21, "Data type correspondences for Fortran", Table 22, "Data type correspondences for MUMPS", Table 23, "Data type correspondences for Pascal", or Table 24, "Data type correspondences for PL/I", respectively. Refer to the two columns of the operative data type correspondences table as the "SQL data type" column and the "host data type column".

c) Any <data type> in the effective SQL parameter list entry of *SR* shall specify a data type listed in the SQL data type column for which the corresponding row in the host data type column is not "None".
NOTE 256 — "Effective SQL parameter list" is defined in Subclause 11.49, "SQL-invoked routine>".

11.51 <drop routine statement>

1. Rationale: Dependency on a user-defined ordering requires only the existence of the ordering.

Replace Syntax Rule 4) with:

4) If RESTRICT is specified, then *SR* shall not be the ordering function included in the descriptor of any user-defined type *UDT*.

2. Rationale: Remove redundant and confusing references to assertion descriptor.

Replace Syntax Rule 5) a) iii) as follows:

5) a) iii) The <search condition> of any constraint descriptor.

3. Rationale: When dropping a routine that is an order function, it is also necessary to perform a <drop user-defined ordering statement>.

Insert the following General Rule:

3.1) If *SR* is the ordering function included in the descriptor of any user-defined type *UDT*, then let *UDTN* be a <user-defined type name> that identifies *UDT*. The following <drop user-defined ordering statement> is effectively executed without further Access Rule checking:

DROP ORDERING FOR *UDTN* CASCADE

11.52 <user-defined cast definition>

1. Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears.

Replace Syntax Rules 4) and 5) with:

4) At least one of *SDT* or *TDT* shall contain a <user-defined type name> or a <reference type>.

5) If *SDT* contains a <user-defined type name>, then let *SSDT* be the schema that includes the descriptor of the user-defined type identified by *SDT*.

2. *Rationale: Provide consistent Syntax Rules for comparison operations.*

Insert the following Syntax Rules:

6) a) i.1) EQUALS ONLY shall be specified.

i.2) The declared type of each attribute of *UDT* shall not be UDT-NC-ordered.

3. *Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears.*

Replace Syntax Rule 7) with:

7) If *TDT* contains a <user-defined type name>, then let *STDT* be the schema that includes the descriptor of the user-defined type identified by *TDT*.

4. *Rationale: Provide consistent Syntax Rules for comparison operations.*

Insert the following Syntax Rule:

8) c) If the result data type of *F* is a large object type, then ORDER FULL shall not be specified.

5. *Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears.*

Replace Syntax Rule 9) with:

9) If both *SDT* and *TDT* contain a <user-defined type name> or a <reference type>, then the <authorization identifier> that owns *SSDT* and the <authorization identifier> that owns *STDT* shall be equivalent.

Replace Syntax Rule 10) c) with:

10) c) The <authorization identifier> that owns *SSDT* or *STDT* (both, if both *SDT* and *TDT* are <user-defined type name>s) shall own the schema that includes the SQL-invoked routine descriptor of *F*.

6. *Rationale: Provide consistent Conformance Rules for comparison operations.*

Insert the following Conformance Rules:

2) Without Feature T042, "Extended LOB data type support", if <state category> is specified, then the declared type of each attribute of *UDT* shall not be LOB-ordered.

3) Without Feature T042, "Extended LOB data type support", if <map category> is specified, the result type of *F* shall not be a large object type.

11.53 <drop user-defined cast statement>

1. *Rationale: Remove redundant and confusing references to assertion descriptor.*

Replace Syntax Rule 7) c) as follows:

7) c) The <search condition> of any constraint descriptor.

2. *Rationale: DROP TRIGGER has no CASCADE capability.*

Replace General Rule 6) with:

6) Let *T* be any trigger whose trigger descriptor includes a trigger action that contains *CS* or *PS*. Let *TN* be the <trigger name> of *T*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER TN
```

11.54 <user-defined ordering definition>

1. *Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears.*

In the Format, replace the production for <user-defined ordering definition> with:

```
<user-defined ordering definition> ::=
     CREATE ORDERING FOR <user-defined type name> <ordering form>
```

Replace Syntax Rule 1) with:

1) Let *UDTN* be the <user-defined type name>. Let *UDT* be the user-defined type identified by *UDTN*.

2. *Rationale: Editorial.*

Replace Syntax Rule 2) with:

2) The descriptor of *UDT* shall include an ordering form that specifies NONE.

3. *Rationale: Incorrect informative note.*

Delete NOTE 264

4. *Rationale: Remove redundant and confusing references to assertion descriptor.*

Replace Syntax Rule 5) c) as follows:

5) c) The <search condition> of any constraint descriptor.

5. Rationale: Provide consistent Syntax Rules for comparison operations.

Insert the following Syntax Rules:

6) i.1) EQUALS ONLY shall be specified.

i.2) The declared type of each attribute of *UDT* shall not be UDT-NC-ordered.

Insert the following Syntax Rule:

8) c) If the result data type of *F* is a large object type, then ORDER FULL shall not be specified.

6. Rationale: Editorial.

Replace Access Rule 1) with:

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema that includes the descriptor of *UDT* and the schema that includes the routine descriptor of *F*.

7. Rationale: SPECIFICTYPE is not a function.

Replace General Rule 1) c) with:

1) c) The following <SQL-invoked routine> is effectively executed:

```
CREATE FUNCTION SNUDT.EQUALS ( UDT1 UDTN, UDT2 UDTN )
  RETURNS BOOLEAN
  SPECIFIC SN
  DETERMINISTIC
  CONTAINS SQL
  STATIC DISPATCH
  RETURN
    ( TRUE AND
    UDT1.SPECIFICTYPE = UDT2.SPECIFICTYPE AND
    UDT1.C1 = UDT2.C1 AND
    .
    .
    .
    UDT1.Cn = UDT2.Cn )
```

8. Rationale: Editorial.

Replace General Rule 4) with:

4) The <specific routine designator> identifying the ordering function, depending on the ordering category, in the descriptor of *UDT* is set to *SRD*.

9. Rationale: Editorial.

Replace Conformance Rule 1) with:

1) Without Feature S251, “User-defined orderings”, conforming SQL shall contain no <user-defined ordering definition>.

10. Rationale: Provide consistent Conformance Rules for comparison operations.

Insert the following Conformance Rules:

2) Without Feature T042, "Extended LOB data type support", if <state category> is specified, then the declared type of each attribute of *UDT* shall not be LOB-ordered.

3) Without Feature T042, "Extended LOB data type support", if <map category> is specified, the result type of *F* shall not be a large object type.

11.55 <drop user-defined ordering statement>

1. Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears.

In the Format, replace the production for <drop user-defined ordering statement> with:

```
<drop user-defined ordering statement> ::=
     DROP ORDERING FOR <user-defined type name> <drop behavior>
```

Replace Syntax Rule 1) with:

1) Let *UDTN* be the <user-defined type name>. Let *UDT* be the user-defined type identified by *UDTN*.

2. Rationale: Editorial.

Replace Syntax Rule 2) with:

2) The descriptor of *UDT* shall include an ordering form that specifies EQUALS or FULL.

3. Rationale: Recognize that a <distinct predicate> may depend on a user-defined ordering. Provide the correct set of dependencies for a user-defined ordering.

Replace Syntax Rule 4) with:

4) Let *S* be the set of types whose comparison type is *UDT*. Let *DEP* designate any of the following:

a) A <comparison predicate>, <between predicate>, <in predicate>, <quantified comparison predicate>, <match predicate>, or <distinct predicate> that simply contains a <row value expression> whose declared type is S-ordered.

b) A <table subquery> simply contained in a <quantified comparison predicate>, <in predicate>, <unique predicate>, or <match predicate> that has a column whose declared type is S-ordered.

c) A <set function specification> that specifies DISTINCT, MAX, or MIN and that simply contains a <value expression> whose declared type is S-ordered.

d) A <group by clause> that contains a <column reference> to a column whose declared type is S-ordered.

e) A <query specification> that specifies the <set quantifier> DISTINCT and that has a column

whose declared type is S-ordered.

f) A <query expression> that specifies or implies UNION DISTINCT and that has a column whose declared type is S-ordered.

g) A <query expression> that specifies INTERSECT or EXCEPT and that has a column whose declared type is S-ordered.

j) A <joined table> that specifies NATURAL or USING and that has a corresponding join column whose declared type is S-ordered.

Replace Syntax Rule 5) with:

5) If RESTRICT is specified, then *DEP* shall not be specified in any of the following:

a) The <SQL routine body> of any routine descriptor.

b) The <query expression> of any view descriptor.

c) The <search condition> of any constraint descriptor or assertion descriptor.

d) The triggered action of any trigger descriptor.

NOTE 267 — If CASCADE is specified, then such referencing objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

4. *Rationale: Provide the correct set of dependencies for a user-defined ordering. DROP TRIGGER has no CASCADE capability.*

Replace General Rules 1), 2), 3), 4), 5) and 6) with:

1) Let *R* be any SQL-invoked routine that contains *DEP* in its <SQL routine body>. Let *SN* be the specific name of *R*. The following <drop routine statement> is effectively executed without further Access Rule checking:
DROP SPECIFIC ROUTINE *SN* CASCADE

2) Let *V* be any view that contains *DEP* in its <query expression>. Let *VN* be the <table name> of *V*. The following <drop view statement> is effectively executed without further Access Rule checking:
DROP VIEW *VN* CASCADE

3) Let *T* be any table that contains *DEP* in the <search condition> of any constraint *C* whose constraint descriptor is included in the table descriptor of *T*. Let *TN* be the <table name> of *T*. Let *TCN* be the <constraint name> of *C*. The following <alter table statement> is effectively executed without further Access Rule checking:
ALTER TABLE *TN* DROP CONSTRAINT *TCN* CASCADE

4) Let *A* be any assertion that contains *DEP* in its <search condition>. Let *AN* be the <constraint name> of *A*. The following <drop assertion statement> is effectively executed without further Access Rule checking:
DROP ASSERTION *AN* CASCADE

5) Let *D* be any domain that contains *DEP* in the <search condition> of any constraint descriptor or in the

<default option> included in the domain descriptor of *D*. Let *DN* be the <domain name> of *D*. The following <drop domain statement> is effectively executed without further Access Rule checking:
DROP DOMAIN *DN* CASCADE

6) Let *T* be any trigger that contains *DEP* in its triggered action. Let *TN* be the <trigger name> of *T*. The following <drop trigger statement> is effectively executed without further Access Rule checking:
DROP TRIGGER *TN*

5. Rationale: Editorial.

Replace General Rule 7) with:

7) In the user-defined data type descriptor of *UDT*, the ordering form is set to NONE and the ordering category is set to STATE. No ordering function is included in the user-defined data type descriptor of *UDT*.

11.56 <transform definition>

1. Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears.

In the Format, replace the production for <transform definition> with:

```
<transform definition> ::=
     CREATE { TRANSFORM | TRANSFORMS } FOR
        <user-defined type name> <transform group>...
```

2. Rationale: The type-preserving property should be required only for to-sql functions that are SQL-invoked methods.

Replace Syntax Rule 4) f) ii) with:

4) f) ii) If *DT* is a structured type and TSF_i is an SQL-invoked method, then TSF_i shall be a type-preserving function.

11.57 <drop transform statement>

1. Rationale: Clarify that implicit schema of user-defined type is schema name of either the <schema definition> or the <SQL-client module definition> in which the corresponding SQL statement appears.

In the Format, replace the production for <drop transform statement> with:

```
<drop transform statement> ::=
        DROP { TRANSFORM | TRANSFORMS } <transforms to be dropped>
           FOR <user-defined type name> <drop behavior>
```

12.2 <grant privilege statement>

1) *Rationale: Correct the references to <grant statement>.*

Replace Syntax Rule 5) with:

5) If the <grant privilege statement> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <object name> shall include the descriptor of *O*. If <grant privilege statement> is contained in a <schema definition> S, then the schema identified by the explicit or implicit qualifier of the <object name> shall include the descriptor of *O* or *S* shall include a <schema element> that creates the descriptor of *O*.

2) *Rationale: Editorial.*

Replace General Rule 7) with:

7) Let *SWH* be the set of privilege descriptors in *CPD* whose action is SELECT WITH HIERARCHY OPTION, and let *ST* be the set of subtables of *O*, then for every grantee *G* in *SWH* and for every table *T* in *ST*, the following <grant statement> is effectively executed without further Access Rule checking:

GRANT SELECT ON *T* TO *G* GRANTED BY *A*

3) *Rationale: Correct the references to <grant statement>.*

Replace Conformance Rule 1) with:

1) Without Feature S024, "Enhanced structured types", a <specific routine designator> contained in a <grant privilege statement> shall not identify a method.

12.4 <select statement: single row>

1) *Rationale: Replace incorrect non-terminal.*

Replace Syntax Rule 1) with:

1) Insert after SR4) For each <target specification> *TS* that is an <SQL variable reference>, the Syntax Rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075-2, apply to *TS* and the corresponding element of the <select list>, as *TARGET* and *VALUE*, respectively.

Replace General Rule 1) with:

1) Insert after GR5) For each <target specification> *TS* that is an <SQL variable reference>, the corresponding value in the row of *Q* is assigned to *TS* according to the General Rules of Subclause 9.2, "Store assignment", in ISO/IEC 9075-2, as *VALUE* and *TARGET*, respectively. The assignment of values to targets in the <select target list> is in an implementation-dependent order.

12.6 <revoke statement>

1. Rationale: Correct the rules for dependency on USAGE privilege for a user-defined type.

Insert the following Syntax Rule:

20) c.1) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in *V* is usage-dependent on *UDT*.

Insert the following Syntax Rule:

22) b.1) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in *TC* is usage-dependent on *UDT*.

Insert the following Syntax Rule:

23) b.1) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in *AX* is usage-dependent on *UDT*.

Insert the following Syntax Rule:

24) c.1) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in any <search condition> of *TR* is usage-dependent on *UDT*.

2. Rationale: There is no <row value expression> immediately contained in a <set clause>.

Replace Syntax Rule 24) j) with:

24) j) SELECT privilege on every <table reference> and <column reference> contained in a <value expression> simply contained in an <update source> contained in the <triggered SQL statement> of *TR*.

3. Rationale: Correct the rules for dependency on USAGE privilege for a user-defined type.

Insert the following Syntax Rule:

24) n.1) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in the <triggered SQL statement> of *TR* is usage-dependent on *UDT*.

4. Rationale: There is no <row value expression> immediately contained in a <set clause>.

Replace Syntax Rule 24) t) with:

24) t) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of any <reference resolution> that is contained in any <value expression> contained in an <update source> contained in the <triggered SQL statement> of *TR*.

5. Rationale: Correct the rules for dependency on USAGE privilege for a user-defined type.

Insert the following Syntax Rule:

25) b.1) USAGE privilege on any user-defined type *UDT* such that some <data type> contained in any <search condition> of *DC* is usage-dependent on *UDT*.

Replace Syntax Rule 27) b) with:

27) b) The revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on any user-defined type *UDT* such that a data type descriptor included in *CD* describes a type that is usage-dependent on *UDT*.

Replace Syntax Rule 29) a) with:

29) a) The revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on any user-defined type *UDT* such that a data type descriptor included in *DT* describes a type that is usage-dependent on *UDT*.

6. *Rationale: Correct the references to schema which do not treat it as a descriptor.*

Replace Syntax Rule 30) with:

30) *S1* is said to be *lost* if the revoke destruction action would result in *A1* no longer having in its applicable privileges USAGE privilege on the default character set included in *S1*.

7. *Rationale: There is no <row value expression> immediately contained in a <set clause>.*

Replace Syntax Rule 35) e) with:

35) e) SELECT privilege on each <table reference> and <column reference> contained in a <value expression> simply contained in an <update source> contained in the <SQL routine body> of *RD*.

8. *Rationale: Correct the rules for dependency on USAGE privilege for a user-defined type.*

Replace Syntax Rule 35) j) with:

35) j) USAGE privilege on each user-defined type *UDT* such that a declared type of any SQL parameter, returns data type, or result cast include in *RD* is usage-dependent on *UDT*.

9. *Rationale: There is no <row value expression> immediately contained in a <set clause>.*

Replace Syntax Rule 35) o) with:

35) o) SELECT privilege WITH HIERARCHY OPTION on at least one supertable of the scoped table of any <reference resolution> that is contained in any <value expression> simply contained in an <update source> contained in the <SQL routine body> of *RD*.

10. *Rationale: Clarify the Access Rules.*

Replace Access Rule 1) with:

1) Case:

a) If the <revoke statement> is a <revoke privilege statement>, then the applicable privileges for *A* shall include a privilege identifying *O*.

b) If the <revoke statement> is a <revoke role statement>, then for every role *R* identified by a <role revoked> the applicable roles of *A* shall include a role *AR* such that there exists a role authorization descriptor with role *R*, grantee *AR*, and the indication that the WITH ADMIN OPTION was granted.

11. Rationale: DROP ASSERTION statement now requires CASCADE.

Replace General Rule 5) with:

5) For every abandoned assertion descriptor *AX*, let *S1.AXN* be the <constraint name> of *AX*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION S1.AXN CASCADE
```

12. Rationale: Correct the references to schema which do not treat it as a descriptor.

Replace General Rule 10) with:

10) For every lost schema *S1*, the default character set of that schema is modified to include the name of the implementation-defined <character set specification> that would have been this schema's default character set had the <schema definition> not specified a <schema character set specification>.

13.1 <SQL-client module definition>

1. Rationale: Editorial - supply missing ellipsis.

In the Format, replace the production <SQL-client module definition> with:

```
<SQL-client module definition> ::=
     <module name clause>
     <language clause>
     <module authorization clause>
     [ <module path specification> ]
     [ <module transform group specification> ]
     [ <temporary table declaration>... ]
     <module contents>...
```

13.2 <module name clause>

1. Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".

Replace Conformance Rule 1) with:

1) Without Feature F461, "Named character sets", <module character set specification> shall not be specified.

13.3 <externally-invoked procedure>

1. Rationale: Complete removal of deprecated feature .

In the Format, replace the production for <host parameter declaration setup> with:

```
<host parameter declaration setup> ::= <host parameter declaration list>
```

Delete NOTE 274.

2. Rationale: The conformance features relative to locators should be defined for module language.

Insert the following Conformance Rules:

1) Without Feature S231, "Structured type locators", a <host parameter data type> that simply contains a <data type> that specifies a structured type shall not contain a <locator indication>.

2) Without Feature S232, "Array locators", a <host parameter data type> that simply contains an <array type> shall not contain a <locator indication>.

13.4 Calls to an <externally-invoked procedure>

1. Rationale: Insert a type definition for BOOLEAN into the Ada package Interface.SQL.

In Syntax Rule 2) e), in the Ada package Interface.SQL, insert a new Ada type declaration immediately following the line reading "**type** DOUBLE_PRECISION is digits dd;":

```
type BOOLEAN is new Boolean;
```

2. Rationale: Add missing SQLSTATE values.

In Syntax Rule 2) e) add the following text to the package definition for SQLSTATE_CODES:

```
DATA_EXCEPTION_INVALID_INTERVAL_FORMAT:
     constant SQLSTATE_TYPE :="22006";
```

In Syntax Rule 2) e) add the following text to the package definition for SQLSTATE_CODES:

```
DATA_EXCEPTION_NULL_VALUE_SUBSTITUTED_FOR_MUTATOR_SUBJECT_PARAMETER:
     constant SQLSTATE_TYPE := "2202D" ;
```

In Syntax Rule 2) e) replace the text in the package definition for SQLSTATE_CODES:

```
WARNING_STATEMENT_TOO_LONG_FOR_INFORMATION_SCHEMA:
     constant SQLSTATE_TYPE :="01005";
```

with:

```
WARNING_STATEMENT_TOO_LONG_FOR_INFORMATION_SCHEMA:
     constant SQLSTATE_TYPE :="0100F";
```

3. *Rationale: Address requirement for multiple diagnostics areas*

In Syntax Rule 2) e) add the following text to the package definition for SQLSTATE_CODES:

```
DIAGNOSTICS_EXCEPTION_NO_SUBCLASS:
     constant SQLSTATE_TYPE := "0Z000"
DIAGNOSTICS_EXCEPTION_MAXIMUM_NUMBER_OF_STACKED_DIAGNOSTICS_AREAS_EXCEEDED:
     constant SQLSTATE_TYPE := "0Z001"
```

4. *Rationale: Remove restrictions on SQLSTATE values generated by external routines*

In Syntax Rule 2) e) delete the following text from the package SQLSTATE_CODES:

```
EXTERNAL_ROUTINE_INVOCATION_EXCEPTION_INVALID_SQLSTATE_RETURNED:
    constant SQLSTATE_TYPE :="39001";
```

5. *Rationale: The SQLSTATE Class codes allocated for SQL/MM no longer needed since they are not used by ISO/IEC 13249.*

In Syntax Rule 2) e) delete the following text from the package definition for SQLSTATE_CODES:

```
SQLMM_PART01_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H1000";
SQLMM_PART02_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H2000";
SQLMM_PART03_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H3000";
SQLMM_PART04_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H4000";
SQLMM_PART05_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H5000";
SQLMM_PART06_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H6000";
SQLMM_PART07_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H7000";
SQLMM_PART08_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H8000";
SQLMM_PART09_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="H9000";
SQLMM_PART10_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="HA000";
SQLMM_PART11_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="HB000";
SQLMM_PART12_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="HC000";
SQLMM_PART13_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="HD000";
SQLMM_PART14_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="HE000";
SQLMM_PART15_NO_SUBCLASS::
    constant SQLSTATE_TYPE :="HF000";
```

6. *Rationale: Insert a General Rule addressing the transfer of Ada Boolean into SQL BOOLEAN sites.*

Insert the following General Rule 2) e) i.0):

2) e) i.0) If the caller language of *EP* is Ada, then if *PI* is False, then a reference to *PN* has the value

false; otherwise, a reference to *PN* has the value *true*.

7. *Rationale: Insert a General Rule addressing the transfer of BOOLEAN data into Ada Boolean parameters.*

Insert the following General Rule 3) g) i.0):

3) g) i.0) If the caller language of *EP* is Ada, then a reference to *PN* that assigns the value *false* to *PN* implicitly assigns the value False to *PI*; a reference to *PN* that assigns the value *true* implicitly assigns the value True to *PI*.

13.5 <SQL procedure statement>

1. *Rationale: Remove redundant item. <grant role statement> is defined in <grant statement>.*

In the Format, replace the production for <SQL schema definition statement> with:

```
<SQL schema definition statement> ::=
      <schema definition>
    | <table definition>
    | <view definition>
    | <SQL-invoked routine>
    | <grant statement>
    | <role definition>
    | <domain definition>
    | <character set definition>
    | <collation definition>
    | <translation definition>
    | <assertion definition>
    | <trigger definition>
    | <user-defined type definition>
    | <user-defined cast definition>
    | <user-defined ordering definition>
    | <transform definition>
```

2. *Rationale: Not all SQL-statements are atomic.*

Insert the following Syntax Rule:

0.1) Let *S* be the <SQL procedure statement>.

3. *Rationale: Correct the definition of possibly non-deterministic.*

Replace Syntax Rule 3) with:

3) An <SQL procedure statement> *S* is *possibly non-deterministic* if and only if at least one of the following is satisfied:

a) *S* is a <select statement: single row> that is possibly non-deterministic.

b) *S* contains a <routine invocation> whose subject routine is an SQL-invoked routine that is possibly non-deterministic.

c) *S* generally contains a <query specification> or a <query expression> that is possibly non-

deterministic.

d) *S* generally contains a <value expression> that is possibly non-deterministic.

4. *Rationale: Clarify the semantics of SQL-data access indication.*

Delete Syntax Rule 4).

Delete Syntax Rule 5).

Delete Syntax Rule 6).

5. *Rationale: Clarify that an executing statement remains an executing statement when it causes another statement to be executed. Not all SQL-statements are atomic.*

Insert the following General Rule:

0.1) Let *S* be the executing statement specified in an application of this Subclause.
NOTE 345 – *S* is necessarily the innermost executing statement of the SQL-session as defined in Subclause 4.39, ''SQL-sessions''.

6. *Rationale: Not all SQL-statements are atomic.*

Replace General Rule 1) with:

1) A statement execution context *NEWSEC* is established for the execution of *S*. Let *OLDSEC* be the most recent execution context. *NEWSEC* becomes the most recent execution context. *NEWSEC* is an atomic execution context, and therefore the most recent atomic execution context, if and only if *S* is an atomic SQL-statement.

7. *Rationale: Not all SQL-statements are atomic.*

Delete General Rule 3).

8. *Rationale: Address requirement for multiple diagnostics areas*

Replace General Rule 5) a) i) 2) with:

5) a) i) 2) The first diagnostics area is emptied.

Replace General Rule 5)a)iii)6) with:

5) a) iii) 6) The first diagnostics area is emptied.

9. *Rationale: Missing General Rule to capture Syntax and Access Rule violations at the time of <SQL procedure statement> execution.*

Insert the following General Rule 5) a) iii) 7.1):

5) a) iii) 7.1) If *S* does not conform to the Syntax Rules and Access Rules of an <SQL procedure statement>, then an exception condition is raised: *syntax error or access rule violation*.

10. Rationale: Address requirement for multiple diagnostics areas

Replace General Rule 5) b) iv) with:

5) b) iv) If *S* is not an <SQL diagnostic statement>, then the first diagnostics area is emptied.

11. Rationale: Simplify confusing rule and avoid the term "re-raise".

Replace General Rule 6) b) ii) 2) with:

6) b) ii) 2) The exception condition with which the execution of *S* completed is raised.

12. Rationale: Address requirement for multiple diagnostics areas

Replace General Rule 7) with:

7) If *S* is not an <SQL diagnostics statement>, then diagnostics information resulting from the execution of *S* is placed into the first diagnostics area, causing the first condition area in the first diagnostics area to become occupied. Whether any other condition areas become occupied is implementation-dependent.

13. Rationale: Not all SQL-statements are atomic.

Insert the following General Rules:

7.1) If *NEWSEC* is atomic, then all savepoints established during its existence are destroyed.

7.2) *NEWSEC* ceases to exist and *OLDSEC* becomes the most recent execution context.

14. Rationale: Clarify that an executing statement remains an executing statement when it causes another statement to be executed.

Insert the following General Rule:

8) *S* ceases to be an executing statement.

NOTE 282.2 — The innermost executing statement, if any, is now the one that was the innermost executing statement that caused *S* to be executed.

15. Rationale: Specify explicitly the implication that F451, "Character set definition" depends on F461, "Named character sets".

Replace Conformance Rules 3) and 10) with:

3) Without Feature F451, “Character set definition”, an <SQL schema definition statement> shall not be a <character set definition>.

10) Without Feature F451, “Character set definition”, an <SQL schema definition statement> shall not be a <drop character set statement>.

13.6 Data type correspondences

1. Rationale: Correct specification of the number of bits in a C character.

In Table 19, replace footnote 2 with:

[2]The length *X* of the character data type corresponding with SQL data type BIT(*L*) is the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits contained in a unit of storage for C **char**.

14.1 <declare cursor>

1. Rationale: Eliminate redundant <collate clause>.

In the Format, replace the production for <sort specification> with:

```
<sort specification> ::= <sort key> [ <ordering specification> ]
```

2. Rationale: Correct non-terminals.

Replace Syntax Rules 7), 8) and 9) with:

7) If <cursor scrollability> is not specified, then NO SCROLL is implicit.

8) If <cursor holdability> is not specified, then WITHOUT HOLD is implicit.

9) If <cursor returnability> is not specified, then WITHOUT RETURN is implicit.

3. Rationale: Prohibit ordering on data types that do not support ordering. Provide consistent Conformance Rules for comparison operations. Provide correct syntactic transformations for expressions in the <order by clause>.

Replace Syntax Rule 18) with:

18) If an <order by clause> is specified, then:

a) Let *OBC* be the <order by clause>. Let *NSK* be the number of <sort specification>s in *OBC*. For each *i* between 1 (one) and *NSK*, let K_i be the <sort key> contained in the *i*-th <sort specification> in *OBC*.

b) Each K_i shall contain a <column reference> and shall not contain a <subquery> or a <set function specification>.

c) The declared type of each K_i shall not be LOB-ordered, array-ordered, UDT-EC-ordered, UDT-NC-ordered, or reference-ordered.

d) If *QE* is a <query expression body> that is a <non-join query expression> that is a <non-join query term> that is a <non-join query primary> that is a <simple table> that is a <query specification>, then the <cursor specification> is said to be a *simple table query*.

e) Case:

i) If <sort specification list> contains any <sort key> K_i that contains a column reference to a column that is not a column of *T*, then:

1) The <cursor specification> shall be a simple table query.

2) Let *TE* be the <table expression> immediately contained in the <query specification> *QS* contained in *QE*.

3) Let *SL* be the <select list> of *QS*. Let *SLT* be obtained from *SL* by replacing each <column reference> with its fully qualified equivalent (i.e., including either an explicit <correlation name> or an explicit <table name> including either the keyword MODULE or an explicit <catalog name> and an explicit <unqualified schema name>; in the case of common column names, each common column name is regarded as fully qualified).

4) Let *OBCT* be obtained from *OBC* by replacing each <column reference> to a column of *TE* with its fully qualified equivalent (i.e., including either an explicit <correlation name> or an explicit <table name> including either the keyword MODULE or an explicit <catalog name> and an explicit <unqualified schema name>; in the case of common column names, each common column name is regarded as fully qualified).

5) For each *i* between 1 (one) and *NSK*, let KT_i be the <sort key> contained in the *i*-th <sort specification> contained in *OBCT*.

6) For each *i* between 1 (one) and *NSK*, if KT_i has the same left normal form derivation as the <value expression> immediately contained in some <derived column> *DC* of *SLT*, then:
NOTE 286.1: "Left normal form derivation" is defined in Subclause 6.1 "Notation" in Part 1 of this international standard.

A) Case:

I) If *DC* simply contains an <as clause>, let *CN* be the <column name> contained in the <as clause>.

II) Otherwise, let *CN* be an implementation-dependent <column name> that is not equivalent to the explicit or implicit <column name> of any other <derived column> contained in *SLT*. Let *VE* be the <value expression> simply contained in *DC*. *DC* is replaced in *SLT* by

VE AS *CN*

B) KT_i is replaced in *OBCT* by

CN

7) Let *SCR* be the set of <column reference>s to columns of *TE* that are remaining in *OBCT* after the preceding transformation.

8) Let *NSCR* be the number of <column reference>s contained in *SCR*. For each *j* between 1 (one) and *NCR*, let C_j be an enumeration of these <column reference>s.

9) Case:

A) If *NSCR* is 0 (zero), then let *SKL* be the zero-length string.

B) Otherwise, let *SKL* be the comma-separated list of <derived column>s

,*C1, C2, . . . , CNCR*

The columns C_j are said to be *extended sort key columns*.

10) Let *ST* be the result of evaluating the <query specification>:

SELECT *SLT SKL TE*

11) Let *EOBC* be *OBCT*.

ii) Otherwise, let *ST* be *T*, and let *EOBC* be *OBC*.

f) *ST* is said to be a *sort table*.

4. *Rationale: A <sort key> of character type need not be a column reference. Standardise terminology.*

Replace Syntax Rule 19) with:

19) If a <sort specification> contains a <collate clause>, then the declared type of the <value expression> contained in the <sort specification> shall be character string. The column descriptor of the corresponding column in the result has the collation specified in <collate clause> and the coercibility characteristic Explicit.

5. *Rationale: Editorial.*

Replace Syntax Rule 20) with:

20) If an <updatability clause> of FOR UPDATE without a <column name list> is specified or implicit, then a <column name list> that consists of the <column name> of every column of *LUT* is implicit.

6. *Rationale: Syntax Rule 22) states the wrong restriction on <sort key>s of user-defined type. The correct restriction is already present in Syntax Rule 18)c).*

Delete Syntax Rule 22).

7. *Rationale: Provide correct syntactic transformations for expressions in the <order by clause>.*

Replace General Rule 2) b) with:

b) Each <sort specification> contained in *EOBC* specifies the sort direction for the corresponding sort key EK_i. If DESC is not specified in the *i*-th <sort specification>, then the sort direction for EK_i is ascending and the applicable <comp op> is the <less than operator>. Otherwise, the sort direction for EK_i is descending and the applicable <comp op> is the <greater than operator>.

8. *Rationale: Clarify that nulls sort as if they were equal. Provide correct syntactic transformations for expressions in the <order by clause>*

Replace General Rule 2) c) with:

2) c) Let *P* be any row of *TS* and let *Q* be any other row of *TS*, and let PV_i and QV_i be the values of EK_i in these rows, respectively. The relative position of rows *P* and *Q* in the result is determined by comparing PV_i and QV_i according to the rules of Subclause 8.2, "<comparison predicate>", where the <comp op> is the applicable <comp op> for EK_i, with the following special treatment of null values. A sort key value that is null is considered equal to another sort key value that is null. Whether a sort key value that is null is considered greater or less than a non-null value is implementation-defined, but all sort key values that are null shall either be considered greater than all non-null values or be considered less than all non-null values. PV_i is said to precede QV_i if the value of the <comparison predicate> "PV_i <comp op> QV_i" is *True* for the applicable <comp op>. If PV_i and QV_i are not null and the result of "PV_i <comp op> QV_i" is *Unknown*, then the relative ordering of PV_i and QV_i is implementation-dependent.

9. Rationale: Use the final <sort specification>.

Replace General Rule 2) d) with:

2) d) In *TS*, the relative position of row *P* is before row *Q* if PV_n precedes QV_n for some *n* greater than 0 (zero) and less than or equal to the number of <sort specification>s and $PV_i = QV_i$ for all $i < n$. The relative order of two rows that are not distinct with respect to the <sort specification>s are implementation-dependent.

10. Rationale: Provide consistent Conformance Rules for comparison operations.

Replace Conformance Rule 9) with:

9) Without Feature S024, "Enhanced structured types", a <value expression> that is a <sort key> shall not be of an ST-ordered declared type.

14.3 <fetch statement>

1. Rationale: Correct use of non-terminals.

Replace Syntax Rule 6) a) i) with:

6) i) If *TS* is an <SQL parameter reference>, then the Syntax Rules of Subclause 9.2, "Store assignment", apply to *TS* and the row type of table *T* as *TARGET* and *VALUE*, respectively.

Replace Syntax Rule 6) b) ii) with:

6) ii) For each <target specification> *TS1* that is an <SQL parameter reference>, the Syntax Rules of Subclause 9.2, "Store assignment", apply to *TS1* and the corresponding column of table *T* as *TARGET* and *VALUE*, respectively.

Replace Syntax Rule 6) b) iii) with:

6) iii) For each <target specification> *TS2* that is a <host parameter specification>, the Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to each *TS2* and the corresponding column of table *T*, as *TARGET* and *VALUE*, respectively.

Replace General Rule 7) a) i) with:

7) i) If *TS* is an <SQL parameter reference>, then the General Rules of Subclause 9.2, "Store assignment", apply to *TS* and the current row as *TARGET* and *VALUE*, respectively.

2. *Rationale: Correct use of non-terminals and symbols.*

Replace General Rule 7) b) i) with:

7) b) i) If *TV* is an <SQL parameter reference>, then the General Rules of Subclause 9.2, ''Store assignment'', apply to *TV* and *SV* as *TARGET* and *VALUE*, respectively.

3. *Rationale: Add missing Syntax and General Rules for <fetch statement>.*

Replace General rule 7) b) ii) with:

7) b) ii) If *TV* is a <host parameter specification>, then the General Rules of Subclause 9.1, "Retrieval assignment" are applied to *TV* and *SV*, as *TARGET* and *VALUE*, respectively.

14.4 <close statement>

1. *Rationale: Closing a cursor does not cause any deferred triggered action to be executed.*

Delete General Rule 5).

14.5 <select statement: single row>

1. *Rationale: Correct use of non-terminals.*

Replace Syntax Rules 3) and 4) with:

3) For each <target specification> *TS* that is an <SQL parameter reference>, the Syntax Rules of Subclause 9.2, "Store assignment", apply to *TS* and the corresponding element of the <select list>, as *TARGET* and *VALUE*, respectively.

4) For each <target specification> *TS* that is a <host parameter specification>, the Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to *TS* and the corresponding element of the <select list>, as *TARGET* and *VALUE*, respectively.

2. *Rationale: Supply the correct definition for non-determinism.*

Delete Syntax Rule 5).

Insert the following Syntax Rule:

6.1) The <select statement: single row> is *possibly non-deterministic* if *S* is possibly non-deterministic.

3. *Rationale: Correct use of non-terminals.*

Replace Syntax Rules 4) and 5) with:

4) For each <target specification> *TS* that is an <SQL parameter reference>, the corresponding value in the row of *Q* is assigned to *TS* according to the General Rules of Subclause 9.2, "Store assignment", as *VALUE* and *TARGET*, respectively. The assignment of values to targets in the <select target list> is in an implementation-dependent order.

5) For each <target specification> *TS* that is a <host parameter specification>, the corresponding value in the row of *Q* is assigned to *TS* according to the General Rules of Subclause 9.1, "Retrieval assignment", as *VALUE* and *TARGET*, respectively. The assignment of values to targets in the <select target list> is in an implementation-dependent order.

14.6 <delete statement: positioned>

1. *Rationale: If ONLY is specified, then parentheses in <target table> are required, otherwise they are optional.*

In the Format, replace the production for <target table> with:

```
<target table> ::=
        <table name>
      | ONLY <left paren> <table name> <right paren>
```

2. *Rationale: Avoid multiple definitions of same symbol and exclude operand cursor from treatment of effect on other cursors.*

Replace General Rule 2) with:

2) If there is any sensitive cursor *SCR*, other than *CR*, that is currently open in the SQL-transaction in which this SQL-statement is being executed, then

   Case:

   a) If *SCR* has not been held into a subsequent SQL-transaction, then either the change resulting from the successful execution of this statement is made visible to *SCR* or an exception condition is raised: *cursor sensitivity exception — request failed.*

   b) Otherwise, whether the change resulting from the successful execution of this SQL-statement is made visible to *SCR* is implementation-defined.

Replace General Rule 3) with:

3) If there is any insensitive cursor *ICR*, other than *CR*, that is currently open, then either the change resulting from the successful execution of this statement is invisible to *ICR*, or an exception condition is raised: *cursor sensitivity exception — request failed.*

3. *Rationale: Use the notion of identical to make specification of effect of <delete statement: positioned> more precise.*

Replace General Rule 7) with:

7) Let *R* be the current row of *CR*. Exactly one row *R1* in *LUT* such that each field in *R* is identical to the corresponding field in *R1* is identified for deletion from *LUT*.

4. *Rationale: Specify the effect on the operand cursor.*

Insert the following General Rule:

7.1) The effect on *CR* is implementation-defined.

5. *Rationale: Use of ONLY requires Feature S111, "ONLY in query expressions".*

Insert the following Conformance Rule:

1) Without Feature S111, "ONLY in query expressions", a <target table> shall not specify ONLY.

14.8 <insert statement>

1. *Rationale: Clean up typed table insertability property for non-instantiable types.*

Replace Syntax Rules 3) and 4) with:

3) For each leaf generally underlying table of *T* whose descriptor includes a user-defined type name *UDTN* the data type descriptor of the user-defined type *UDT* identified by *UDTN* shall indicate that *UDT* is instantiable.

2. *Rationale: Remove a syntactic ambiguity, that VALUES (1) may be parsed either as <insert columns and source> ::= <from subquery> ::= <query expression> ::= <table value constructor> ::= VALUES (1) or <insert columns and source> ::= <from constructor> ::= <contextually typed table value constructor> ::= VALUES (1).*

Insert the following Syntax Rule:

8.1) A <query expression> simply contained in a <from subquery> shall not be a <table value constructor>.

3. *Rationale: Use the correct non-terminal symbols.*

Replace Syntax Rule 9) with:

9) An <insert columns and source> that specifies DEFAULT VALUES is implicitly replaced by an <insert columns and source> that specifies a <contextually typed table value constructor> of the form

```
VALUES (DEFAULT, DEFAULT, . . . , DEFAULT)
```

where the number of "DEFAULT" entries is equal to the number of columns of T.

4. Rationale: Cater for self-referencing column in underlying table of T.

Replace General Rules 7) b), c) and d) with:

7) b) If *T* has a column *RC* of which some underlying column is a self-referencing column, then

Case:

i) If *RC* is a system-generated self-referencing column, then the value of *RC* is effectively replaced by the REF value of the candidate row.

ii) If *RC* is a derived self-referencing column, then the value of *RC* is effectively replaced by a value derived from the columns in the candidate row that correspond to the list of attributes of the derived representation of the reference type of *RC* in an implementation-dependent manner.

c) For each object column in the candidate row, let C_i be the object column identified by the *i*-th <column name> in the <insert column list> and let SV_i be the *i*-th value of *R*.

d) For every C_i for which one of the following conditions is true:

i) C_i is not a self-referencing column of *T;*

ii) C_i is a user-generated self-referencing column of *T;*

iii) C_i is a self-referencing column of *T* and OVERRIDING SYSTEM VALUE is specified;

the General Rules of Subclause 9.2, "Store assignment", are applied to C_i and SV_i as *TARGET* and *SOURCE*, respectively.

14.9 <update statement: positioned>

1. Rationale: Remove undefined SET ROW functionality.

In the Format, replace the production for <update target> with:

```
<update target> ::=
      <object column>
    | <object column>
        <left bracket or trigraph>
        <simple value specification>
        <right bracket or trigraph>
```

2. Rationale: Replace imprecise wording with precise wording.

Replace Syntax Rule 8) with:

8) If *CR* is an ordered cursor, then for each <object column> *OC* contained in <set clause list>, the <order by clause> of the defining <cursor specification> for *CR* shall not generally contain a <column reference> that references *OC* or an underlying column of the column identified by *OC*.

3. *Rationale: Remove undefined SET ROW functionality.*

Delete Syntax Rule 9).

4. *Rationale: permit <set function specification> within a <subquery> within a <set clause>.*

Replace Syntax Rule 12) with:

12) A <value expression> simply contained in an <update source> in a <set clause> shall not directly contain a <set function specification>.

5. *Rationale: There is not necessarily a <row value expression> in <set clause>*

Replace Syntax Rule 13) c) with:

13) c) Let $RCVE_i$, 1 (one) ≤ i ≤ N, be the <update source> simply contained in the i-th <set clause> MSC_i that contains a <mutated set clause>. Let FN_i be the <method name> immediately contained in MSC_i. Let MT_i be the <mutated target> immediately contained in MSC_i.

Replace Syntax Rule 13) d) iii) 1) with:

13) d) iii) 1) Every occurrence of a column reference that refers to C in SC_i is replaced by the <update source> contained in SC_j.

6. *Rationale: Avoid multiple definitions of same symbol and exclude operand cursor from treatment of effect on other cursors.*

Replace General Rule 2) with:

2) If there is any sensitive cursor *SCR*, other than *CR*, that is currently open in the SQL-transaction in which this SQL-statement is being executed, then

Case:

a) If *SCR* has not been held into a subsequent SQL-transaction, then either the change resulting from the successful execution of this statement is made visible to *SCR* or an exception condition is raised: *cursor sensitivity exception — request failed.*

b) Otherwise, whether the change resulting from the successful execution of this SQL-statement is made visible to *SCR* is implementation-defined.

Replace General Rule 3) with:

3) If there is any insensitive cursor *ICR*, other than *CR*, that is currently open, then either the change resulting from the successful execution of this statement is invisible to *ICR*, or an exception condition is raised: *cursor sensitivity exception — request failed.*

7. *Rationale: Replace reference to non-existent BNF term.*

Replace General Rule 12) with:

12) A <set clause> specifies one or more object columns and an update value. An *object column* is a

column identified by an <object column> in the <set clause>. The update value is the value specified by the <update source> contained in the <set clause>.
NOTE 307 – The data values allowable in the current row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 14.22, ''Effect of replacing some rows in a viewed table''.

8. Rationale: Remove undefined SET ROW functionality.

Replace General Rule 15) with:

15) Let *CL* be the columns of *T* identified by the <object columns> contained in the <set clause list>.

9. Rationale: Use the notion of identical to make specification of <update statement: positioned> more precise.

Replace General Rule 16) with:

16) Let *R1* be the candidate new row. The current row *R* of *CR* is replaced by *R1*. Exactly one row *TR* in *T* such that each field in *R* is identical to the corresponding field in *TR* is identified for replacement in *T*. Let *TR1* be a row consisting of the fields of *R1* and the fields of *TR* that have no corresponding fields in *R1*, ordered according to the order of their corresponding columns in *T*. *TR1* is the replacement row for *TR* and { (*TR*, *TR1*) } is the replacement set for *T*.

10. Rationale: Specify the effect on the operand cursor.

Insert the following General Rule:

16.1) The effect on *CR* is implementation-defined.

11. Rationale: Remove undefined SET ROW functionality.

Delete Conformance Rule 3).

14.10 <update statement: searched>

1. Rationale: Remove undefined SET ROW functionality.

Delete Syntax Rule 4).

2. Rationale: permit <set function specification> within a <subquery> within a <set clause>.

Replace Syntax Rule 5) with:

5) A <value expression> simply contained in an <update source> in a <set clause> shall not directly contain a <set function specification>.

3. Rationale: There is not necessarily a <row value expression> in <set clause>

Replace Syntax Rule 7) c) with:

7) c) Let $RCVE_i$, 1 (one) ≤ i ≤ N, be the <update source> simply contained in the i-th <set clause> MSC_i that contains a <mutated set clause>. Let FN_i be the <method name> immediately contained in

MSC_i. Let MT_i be the <mutated target> immediately contained in MSC_i.

Replace Syntax Rule 7) d) iii) 1) with:

7) d) iii) 1) Every occurrence of a column reference that refers to C in SC_i is replaced by the <update source> contained in SC_j

4. Rationale: Remove undefined SET ROW functionality.

Replace General Rule 13) with:

13) Let *CL* be the columns of *T* identified by the <object columns> contained in the <set clause list>.

14.11<temporary table declaration>

1. Rationale: Correct privileges for <temporary tables>s.

Replace General Rule 4) with:

4) A set of privilege descriptors is created that define the privileges INSERT, SELECT, UPDATE, DELETE, and REFERENCES on this table and INSERT, SELECT, UPDATE, and REFERENCES for every <column definition> in the table definition to *A*. These privileges are not grantable. The grantee is "PUBLIC".

14.12 <free locator statement>

1. Rationale: Add missing rule.

Insert the following Conformance Rule:

1) Without Feature T561, "Holdable locators", conforming SQL language shall not contain any <free locator statement>.

14.14 Effect of deleting rows from base table

1. Rationale: Provide the rule of initialization of transition table, association between subject table of trigger and transition table on each trigger context and construction of the value of transition variable.

Insert the following General Rule:

4.1) For every table *ST*, its old transition table is initialized as empty table and associated with *ST* on *CTEC*.

4.2) For every row that is indicated for deletion, the value of its old transition variable is constructed by copying that row.

2. *Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.*

Replace General Rule 7) with:

7) Each row that is marked for deletion from *T* is deleted from *T*. On each deletion process of row, the old transition variable is inserted into old transition table associated with *F* on *CTEC*.
NOTE 397 – See Subclause 4.15.3, ''Operations involving tables'', for the effect of deleting a row from a table.

3. *Rationale: Provide the rule of destruction of transition table.*

Insert the following General Rule:

8.1) For every subject table in *SSC*, the transition table associated with that table is destroyed.

14.17 Effect of inserting tables into base tables

1. *Rationale: Provide the rule of initialization of transition table, association between subject table of trigger and transition table on each trigger context and construction of the value of transition variable.*

Insert the following General Rule:

4.1) For every table *ST*, its new transition table is initialized as empty table and associated *F* on *CTEC*.

4.2) For every row that is indicated for insertion, the value of its new transition variable is constructed by copying its candidate row.

2. *Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.*

Insert the following General Rule:

7) d) On each insertion process of row, the new transition variable is inserted into new transition table associated *F* on *CTEC*.

3. *Rationale: Provide the rule of destruction of transition table.*

Insert the following General Rule:

8.1) For every subject table in *SSC*, the transition table associated with that table is destroyed.

14.18 Effect of inserting a table into a derived table

1. Rationale: Editorial.

Replace General Rule 2) b) iii) with:

2) b) iii) The candidate rows are added to the corresponding S_i.

14.20 Effect of replacing rows from base table

1. Rationale: Provide the rule of initialization of transition table, association between subject table of trigger and transition table on each trigger context and construction of the value of transition variable.

Insert the following General Rule:

5.1) For every table *ST*, its old transition table and new transition table is initialized as empty table and associated *F* on *CTEC*.

5.2) For every row that is indicated for replacement, the value of its old transition variable is constructed by copying that row, and the value of its new transition variable is constructed by copying its candidate row.

2. Rationale: Provide the rule of replacement of row with new transition variable and insertion of the value of transition variable.

Replace General Rule 8) with:

8) For every table *T* in *TT*, for every table *ST* that is a supertable or a subtable of *T*, for every row *R* that is identified for replacement in *ST*, *R* is replaced by its new transition variable. *R* is no longer identified for replacement. *ST* is no longer identified for replacement processing. On each replacement process of row, the old transition variable is inserted into old transition table associated *F* on *CTEC* and the new transition variable is inserted into new transition table associated *F* on *CTEC*.
NOTE 403 – The General Rules of Subclause 11.8, ''<referential constraint definition>'', are now applicable.

3. Rationale: Provide the rule of destruction of transition table.

Insert the following General Rule:

9.1) For every subject table in *SSC*, the transition table associated with that table is destroyed.

15.2 <return statement>

1. Rationale: Move the check for the most specific type of the returned value to <routine invocation>. Delete redundant invocation of store assignment rules.

Replace General Rules 1), 2) and 3) with:

1) The value of *VE* is the *returned value* of the execution of the SQL routine body of *F*.

16.4 <savepoint statement>

1. Rationale: Remove the anomaly in <savepoint specifier>.

In the Format, replace the production for <savepoint specifier> with:

```
<savepoint specifier> ::= <savepoint name>
```

Delete Syntax Rule 1)

Replace General Rules 1) and 2) with:

1) Let *S* be the <savepoint name>.

Delete General Rule 5)

16.5 <release savepoint statement>

1. Rationale: Remove the anomaly in <savepoint specifier>.

Delete Syntax Rule 1)

Replace General Rule 1) with:

1) Let *S* be the <savepoint name>.

16.6 <commit statement>

1. Rationale: Uniformity of terminology.

Replace General Rule 9) with:

9) The current SQL-transaction is terminated. If AND CHAIN was specified, then a new SQL-transaction is initiated with the same access mode, isolation level, and diagnostics area limit as the SQL-transaction just terminated. Any branch transactions of the SQL-transaction are initiated with the same access mode, isolation level, and diagnostics area limit as the corresponding branch of the SQL-transaction just terminated.

16.7 <rollback statement>

1. Rationale: Remove the anomaly in <savepoint specifier>.

Delete Syntax Rule 1)

Replace General Rule 3) a) with:

3) a) Let *S* be the <savepoint name>.

2. *Rationale: Uniformity of terminology.*

Replace General Rule 2) f) with:

2) f) The current SQL-transaction is terminated. If AND CHAIN was specified, then a new SQL-transaction is initiated with the same access mode, isolation level, and diagnostics area limit as the SQL-transaction just terminated. Any branch transactions of the SQL-transaction are initiated with the same access mode, isolation level, and diagnostics area limit as the corresponding branch of the SQL-transaction just terminated.

3. *Rationale: Correct the specification of which locators are marked invalid when an SQL-transaction ends.*

Replace General Rule 3) f) with:

3) f) Every valid locator generated in the current SQL-transaction subsequent to the establishment of *S* is marked invalid.

19.1 <get diagnostics statement>

1. *Rationale: Address requirement for multiple diagnostics areas*

Replace the body of the **Function** with:

Get exception or completion condition information from a diagnostics area.

2. *Rationale: Inconsistent data type for PARAMETER_MODE.*

In Table 25, Identifiers for use with <get diagnostics statement>, replace the row for PARAMETER_MODE with:

<identifier>	Declared Type
<condition information item name>s	
PARAMETER_MODE	variable-length character string with maximum length 5

3. *Rationale: Address requirement for multiple diagnostics areas*

Insert the following General Rule:

0.1) Let *DA* be the first diagnostics area.

Replace General Rule 1) with:

1) Specification of <statement information item> assigns the value of the specified statement information item in *DA* to <simple target specification>.

a) The value of NUMBER is the number of exception or completion conditions that have been stored in *DA* as a result of executing the previous SQL-statement other than a <get diagnostics statement>.
NOTE 337 – The <get diagnostics statement> itself may return information via the SQLSTATE parameter, but does not modify the previous contents of *DA*.

b) The value of MORE is:

Y More conditions were raised during execution of the SQL-statement than there are condition areas in *DA*.

N All of the conditions that were raised during execution of the SQL-statement have been stored in *DA*.

c) The value of COMMAND_FUNCTION is the identification of the SQL-statement executed. Table 26, ''SQL-statement codes'' specifies the identifier of the SQL-statements.

d) The value of COMMAND_FUNCTION_CODE is a number identifying the SQL-statement executed. Table 26, ''SQL-statement codes'' specifies the code for the SQL-statements. Positive values are reserved for SQL-statements defined by ISO/IEC 9075; negative values are reserved for implementation-defined SQL-statements.

e) The value of ROW_COUNT is the number of rows affected as the result of executing a <delete statement: searched>, <insert statement>, or <update statement: searched> or as a direct result of executing the previous SQL-statement. Let *S* be the <delete statement: searched>, <insert statement>, or <update statement: searched>. Let *T* be the table identified by the <table name> directly contained in *S*.

Case:

i) If *S* is an <insert statement>, then the value of ROW_COUNT is the number of rows inserted into *T*.

ii) If *S* is not an <insert statement> and does not contain a <search condition>, then the value of ROW_COUNT is the cardinality of *T* before the execution of *S*.

iii) Otherwise, let *SC* be the <search condition> directly contained in *S*. The value of ROW_COUNT is effectively derived by executing the statement:

```
SELECT COUNT(*) FROM T WHERE SC
```

before the execution of *S*.

The value of ROW_COUNT following the execution of an SQL-statement that does not directly result in the execution of a <delete statement: searched>, an <insert statement>, or an <update statement: searched> is implementation-dependent

f) The value of TRANSACTIONS_COMMITTED is the number of SQL-transactions that have been committed since the most recent time at which *DA* was emptied.

g) The value of TRANSACTIONS_ROLLED_BACK is the number of SQL-transactions that have been rolled back since the most recent time at which *DA* was emptied.

h) The value of TRANSACTION_ACTIVE is 1 (one) if an SQL-transaction is currently active, and 0 (zero) if an SQL-transaction is not currently active.
NOTE 341 – TRANSACTION_ACTIVE indicates whether an SQL-transaction is active upon return from an external routine.

Replace General Rule 2) with:

2) If <condition information> is specified, then let *N* be the value of <condition number>. If *N* is less than 1 (one) or greater than the number of occupied condition areas in *DA*, then an exception condition is raised: *invalid condition number*. If <condition number> has the value 1, then the diagnostics information retrieved corresponds to the condition indicated by the SQLSTATE value actually returned by execution of the previous SQL-statement other than a <get diagnostics statement>. Otherwise, the association between <condition number> values and specific conditions raised during evaluation of the General Rules for that SQL-statement is implementation-dependent.

Replace the 1st sentence of General Rule 3) with:

Specification of <condition information item> assigns the value of the specified condition information item in the *N*-th condition area in *DA* to <simple target specification>.

4. Rationale: Editorial. Non-reserved word - DYNAMIC_FUNCTION - not defined. Only defined in Part 5.

Replace General Rule 3) p) with:

3) p) The values of CONNECTION_NAME and SERVER_NAME are respectively

Case:

i) If COMMAND_FUNCTION identifies an <SQL connection statement>, then the <connection name> and the <SQL-server name> specified by or implied by the <SQL connection statement>.

ii) Otherwise, the <connection name> and <SQL-server name> of the SQL-session in which the condition was raised.

5. Rationale: Delete statement codes for declarations.

Delete the following rows in Table 26, "SQL-statement codes":

SQL-statement	Identifier	Code
<declare cursor>	DECLARE CURSOR	101

6. Rationale: Correct the classification of SQL-statements.

Replace the following rows in Table 26, "SQL-statement codes":

SQL-statement	Identifier	Code
<grant statement>	GRANT	48
<revoke statement>	REVOKE	59

with:

SQL-statement	Identifier	Code
<grant privilege statement>	GRANT	48
<revoke privilege statement>	REVOKE	59
<revoke role statement>	REVOKE ROLE	129

7. Rationale: Correct the classification of SQL-statements.

Insert the following rows into Table 26, "SQL-statement codes":

SQL-statement	Identifier	Code
<user-defined cast definition>	CREATE CAST	52
<drop-user-defined cast definition>	DROP CAST	78

19.2 Pushing and popping the diagnostics area stack

1. Rationale: Address requirement for multiple diagnostics areas

Insert the following Subclause

19.2 Pushing and popping the diagnostics area stack

Function

Define operations on the diagnostics area stack

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *OP* be the *OPERATION* and let *DAS* be the *STACK* specified in an application of this Subclause.

2) Case:

a) If *OP* is PUSH, then

Case:

i) If the number of diagnostics areas in *DAS* is equal to the implementation-dependent maximum number of diagnostics areas per diagnostics area stack, then an exception is raised: *maximum number of stacked diagnostics areas exceeded*.

ii) Otherwise, *DAS* is pushed and the contents of the second diagnostics area in *DAS* are copied to the first.

b) If *OP* is POP, then the first diagnostics area is removed from *DAS* such that all subsequent diagnostics areas in *DAS* move up one position, the second becoming the first, the third the second and so on.

20.4 CARDINAL_NUMBER domain

1. *Rationale: Remove undefined terming in the description.*

Replace Description 1) with:

1) The domain CARDINAL_NUMBER contains any non-negative number that is less than or equal to the implementation-defined maximum for INTEGER.

20.9 APPLICABLE_ROLES view

1. *Rationale: Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.*

In the Definition, replace the view definition with:

```
CREATE RECURSIVE VIEW APPLICABLE_ROLES
           ( GRANTEE, ROLE_NAME, IS_GRANTABLE ) AS
  ( ( SELECT GRANTEE, ROLE_NAME, IS_GRANTABLE
       FROM DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS
      WHERE ( GRANTEE IN ( CURRENT_USER, 'PUBLIC' )
              OR
              GRANTEE IN ( SELECT ROLE_NAME
                             FROM ENABLED_ROLES )
            )
    )
    UNION
    ( SELECT RAD.GRANTEE, RAD.ROLE_NAME, RAD.IS_GRANTABLE
        FROM DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
             JOIN APPLICABLE_ROLES R
               ON RAD.GRANTEE = R.ROLE_NAME
    )
  )
;
```

20.11 ATTRIBUTES view

1. *Rationale: Add missing column to the definition of the ATTRIBUTES view. Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.*

In the Definition, replace the view definition with:

```
CREATE VIEW ATTRIBUTES AS
SELECT DISTINCT
       UDT_CATALOG, UDT_SCHEMA, UDT_NAME,
       A.ATTRIBUTE_NAME, ORDINAL_POSITION,
       ATTRIBUTE_DEFAULT,
       IS_NULLABLE, DATA_TYPE, CHARACTER_MAXIMUM_LENGTH,
       CHARACTER_OCTET_LENGTH, C1.CHARACTER_SET_CATALOG,
       C1.CHARACTER_SET_SCHEMA, C1.CHARACTER_SET_NAME,
       D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA, D1.COLLATION_NAME,
       NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
       DATETIME_PRECISION, INTERVAL_TYPE, INTERVAL_PRECISION,
```

```
        D1.USER_DEFINED_TYPE_CATALOG AS ATTRIBUTE_UDT_CATALOG,
        D1.USER_DEFINED_TYPE_SCHEMA AS ATTRIBUTE_UDT_SCHEMA,
        D1.USER_DEFINED_TYPE_NAME AS ATTRIBUTE_UDT_NAME,
        D1.SCOPE_CATALOG, D1.SCOPE_SCHEMA, D1.SCOPE_NAME,
        MAXIMUM_CARDINALITY, A.DTD_IDENTIFIER, CHECK_REFERENCES,
        IS_DERIVED_REFERENCE_ATTRIBUTE
    FROM ( DEFINITION_SCHEMA.ATTRIBUTES AS A
           LEFT JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
                       LEFT JOIN DEFINITION_SCHEMA.COLLATIONS AS C1
                         ON ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA,
                                C1.COLLATION_NAME )
                                = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA,
                                    D1.COLLATION_NAME ) ) )
             ON ( ( A.UDT_CATALOG, A.UDT_SCHEMA, A.UDT_NAME,
                    'USER-DEFINED TYPE', A.DTD_IDENTIFIER )
                  = ( D1.OBJECT_CATALOG, D1.OBJECT_SCHEMA, D1.OBJECT_NAME,
                      D1.OBJECT_TYPE, D1.DTD_IDENTIFIER ) ) )
    WHERE ( A.UDT_CATALOG, A.UDT_SCHEMA, A.UDT_NAME )
          IN ( SELECT TYPE_SCHEMA, UDTP.USER_DEFINED_TYPE,
                      UDTP.USER_DEFINED_TYPE_CATALOG, UDTP.USER_DEFINED__NAME
                 FROM DEFINITION_SCHEMA.USER_DEFINED_TYPE_PRIVILEGES AS UDTP
                WHERE ( UDTP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                        OR
                        UDTP.GRANTEE IN ( SELECT ROLE_NAME
                                            FROM ENABLED_ROLES ) ) )
          AND
          A.UDT_CATALOG = ( SELECT CATALOG_NAME
                              FROM INFORMATION_SCHEMA_CATALOG_NAME );
```

20.12 CHARACTER_SETS view

1. *Rationale: Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.*

In the Definition, replace the <where clause> with:

```
    WHERE ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
            'CHARACTER SET' )
          IN ( SELECT UP.OBJECT_CATALOG, UP.OBJECT_SCHEMA, UP.OBJECT_NAME,
                      UP.OBJECT_TYPE
                 FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES AS UP
                WHERE ( UP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                        OR
                        UP.GRANTEE IN ( SELECT ROLE_NAME
                                          FROM ENABLED_ROLES ) ) )
          AND
          CHARACTER_SET_CATALOG = ( SELECT CATALOG_NAME
                                      FROM INFORMATION_SCHEMA_CATALOG_NAME );
```

20.12a CHECK_CONSTRAINT_ROUTINE_USAGE view

1. Rationale: Add missing tables to describe the usage of routines by other schema objects.

Insert the following Subclause:

20.12a CHECK_CONSTRAINT_ROUTINE_USAGE view

Function

Identify each SQL-invoked routine owned by a given user on which a domain constraint, table check constraint or assertion defined in this catalog is dependent.

Definition

```
CREATE VIEW CHECK_CONSTRAINT_ROUTINE_USAGE AS
  SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
         SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME
    FROM ( DEFINITION_SCHEMA.CHECK_CONSTRAINT_ROUTINE_USAGE
           JOIN DEFINITION_SCHEMA.CHECK_CONSTRAINTS
             USING ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA,
                     CONSTRAINT_NAME ) )
           JOIN DEFINITION_SCHEMA.SCHEMATA S
             ON ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA )
                = ( S.CATALOG_NAME, S.SCHEMA_NAME )
   WHERE ( SCHEMA_OWNER = CURRENT_USER
           OR
           SCHEMA_OWNER IN ( SELECT ROLE_NAME
                               FROM ENABLED_ROLES ) )
         AND
         SPECIFIC_CATALOG = ( SELECT CATALOG_NAME
                                FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE CHECK_CONSTRAINT_ROUTINE_USAGE
  TO PUBLIC WITH GRANT OPTION;
```

Conformance rules

1) Without Feature F341, "Usage tables", conforming SQL language shall not reference INFORMATION_SCHEMA.CHECK_CONSTRAINT_ROUTINE_USAGE.

2) Without Feature F391, "Long identifiers", conforming SQL language shall not reference INFORMATION_SCHEMA.CHECK_CONSTRAINT_ROUTINE_USAGE.

20.14 COLLATION view

1. Rationale: Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.

In the Definition, replace the <where clause> with:

```
WHERE ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, 'COLLATION' )
      IN ( SELECT UP.OBJECT_CATALOG, UP.OBJECT_SCHEMA, UP.OBJECT_NAME,
                  UP.OBJECT_TYPE
```

```
                FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES AS UP
               WHERE ( UP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                       OR
                       UP.GRANTEE IN ( SELECT ROLE_NAME
                                         FROM ENABLED_ROLES )
                     )
           )
         AND
         COLLATION_CATALOG = ( SELECT CATALOG_NAME
                                 FROM INFORMATION_SCHEMA_CATALOG_NAME );
```

20.16 COLUMN_PRIVILEGES view

1. *Rationale: Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.*

In the Definition, replace the <where clause> with:

```
WHERE ( GRANTEE IN ( 'PUBLIC', CURRENT_USER )
        OR
        GRANTEE IN ( SELECT ROLE_NAME
                       FROM ENABLED_ROLES )
        OR
        GRANTOR = CURRENT_USER
        OR
        GRANTOR IN ( SELECT ROLE_NAME
                       FROM ENABLED_ROLES )
      )
      AND
      TABLE_CATALOG = ( SELECT CATALOG_NAME
                          FROM INFORMATION_SCHEMA_CATALOG_NAME )
;
```

20.17 COLUMN_UDT_USAGE view

1. *Rational: The columns USER_DEFINED_TYPE_CATALOG and USER_DEFINED_TYPE_SCHEMA are not defined for table DEFINITION_SCHEMA.SCHEMATA.*

Replace the Definition with:

```
CREATE VIEW COLUMN_UDT_USAGE AS
  SELECT D.USER_DEFINED_TYPE_CATALOG AS UDT_CATALOG,
         D.USER_DEFINED_TYPE_SCHEMA AS UDT_SCHEMA,
         D.USER_DEFINED_TYPE_NAME AS UDT_NAME,
         C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME, C.COLUMN_NAME
    FROM ( DEFINITION_SCHEMA.COLUMNS AS C
           JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D
                  JOIN DEFINITION_SCHEMA.SCHEMATA AS S
                    ON ( D.USER_DEFINED_TYPE_CATALOG,
                         D.USER_DEFINED_TYPE_SCHEMA )
                       = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
             ON ( ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME,
                    'TABLE', C.DTD_IDENTIFIER )
                  = ( D.OBJECT_CATALOG, D.OBJECT_SCHEMA, D.OBJECT_NAME,
                      D.OBJECT_TYPE, D.DTD_IDENTIFIER ) ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
            OR
```

```
            S.SCHEMA_OWNER IN ( SELECT R.ROLE_NAME
                                  FROM ENABLED_ROLES AS R) )
          AND
          D.DATA_TYPE = 'USER-DEFINED';

GRANT SELECT ON TABLE COLUMN_UDT_USAGE
  TO PUBLIC WITH GRANT OPTION;
```

20.18 COLUMNS view

1. *Rationale: Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.*

Replace the Definition with:

```
CREATE VIEW COLUMNS AS
SELECT DISTINCT
       TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
       C.COLUMN_NAME, ORDINAL_POSITION,
       COLUMN_DEFAULT,
       IS_NULLABLE,
       COALESCE ( D1.DATA_TYPE, D2.DATA_TYPE ) AS DATA_TYPE,
       COALESCE ( D1.CHARACTER_MAXIMUM_LENGTH, D2.CHARACTER_MAXIMUM_LENGTH )
         AS CHARACTER_MAXIMUM_LENGTH,
       COALESCE ( D1.CHARACTER_OCTET_LENGTH, D2.CHARACTER_OCTET_LENGTH )
         AS CHARACTER_OCTET_LENGTH,
       COALESCE ( D1.NUMERIC_PRECISION, D2.NUMERIC_PRECISION )
         AS NUMERIC_PRECISION,
       COALESCE ( D1.NUMERIC_PRECISION_RADIX, D2.NUMERIC_PRECISION_RADIX )
         AS NUMERIC_PRECISION_RADIX,
       COALESCE ( D1.NUMERIC_SCALE, D2.NUMERIC_SCALE )
         AS NUMERIC_SCALE,
       COALESCE ( D1.DATETIME_PRECISION, D2.DATETIME_PRECISION )
         AS DATETIME_PRECISION,
       COALESCE ( D1.INTERVAL_TYPE, D2.INTERVAL_TYPE )
         AS INTERVAL_TYPE,
       COALESCE ( D1.INTERVAL_PRECISION, D2.INTERVAL_PRECISION )
         AS INTERVAL_PRECISION,
       COALESCE ( C1.CHARACTER_SET_CATALOG, C2.CHARACTER_SET_CATALOG )
         AS CHARACTER_SET_CATALOG,
       COALESCE ( C1.CHARACTER_SET_SCHEMA, C2.CHARACTER_SET_SCHEMA )
         AS CHARACTER_SET_SCHEMA,
       COALESCE ( C1.CHARACTER_SET_NAME, C2.CHARACTER_SET_NAME )
         AS CHARACTER_SET_NAME,
       COALESCE ( D1.COLLATION_CATALOG, D2.COLLATION_CATALOG )
         AS COLLATION_CATALOG,
       COALESCE ( D1.COLLATION_SCHEMA, D2.COLLATION_SCHEMA )
         AS COLLATION_SCHEMA,
       COALESCE ( D1.COLLATION_NAME, D2.COLLATION_NAME )
         AS COLLATION_NAME,
       DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
       COALESCE ( D1.USER_DEFINED_TYPE_CATALOG, D2.USER_DEFINED_TYPE_CATALOG )
         AS UDT_CATALOG,
       COALESCE ( D1.USER_DEFINED_TYPE_SCHEMA, D2.USER_DEFINED_TYPE_SCHEMA )
         AS UDT_SCHEMA,
       COALESCE ( D1.USER_DEFINED_TYPE_NAME, D2.USER_DEFINED_TYPE_NAME )
         AS UDT_NAME,
       COALESCE ( D1.SCOPE_CATALOG, D2.SCOPE_CATALOG ) AS SCOPE_CATALOG,
       COALESCE ( D1.SCOPE_SCHEMA, D2.SCOPE_SCHEMA ) AS SCOPE_SCHEMA,
       COALESCE ( D1.SCOPE_NAME, D2.SCOPE_NAME ) AS SCOPE_NAME,
```

```
        COALESCE ( D1.MAXIMUM_CARDINALITY, D2.MAXIMUM_CARDINALITY )
          AS MAXIMUM_CARDINALITY,
        COALESCE ( D1.DTD_IDENTIFIER, D2.DTD_IDENTIFIER ) AS DTD_IDENTIFIER,
        IS_SELF_REFERENCING
    FROM ( DEFINITION_SCHEMA.COLUMNS AS C
          LEFT JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
                      LEFT JOIN DEFINITION_SCHEMA.COLLATIONS AS C1
                        ON ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA,
                               C1.COLLATION_NAME )
                             = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA,
                                 D1.COLLATION_NAME ) ) )
            ON ( ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME,
                   'TABLE', C.DTD_IDENTIFIER )
                 = ( D1.OBJECT_CATALOG, D1.OBJECT_SCHEMA, D1.OBJECT_NAME,
                     D1.OBJECT_TYPE, D1.DTD_IDENTIFIER ) ) )
        LEFT JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D2
                    LEFT JOIN DEFINITION_SCHEMA.COLLATIONS AS C2
                      ON ( ( C2.COLLATION_CATALOG, C2.COLLATION_SCHEMA,
                             C2.COLLATION_NAME )
                           = ( D2.COLLATION_CATALOG, D2.COLLATION_SCHEMA,
                               D2.COLLATION_NAME ) ) )
          ON ( ( C.DOMAIN_CATALOG, C.DOMAIN_SCHEMA, C.DOMAIN_NAME,
                 'DOMAIN', C.DTD_IDENTIFIER )
               = ( D2.OBJECT_CATALOG, D2.OBJECT_SCHEMA, D2.OBJECT_NAME,
                   D2.OBJECT_TYPE, D2.DTD_IDENTIFIER ) )
  WHERE ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME, C.COLUMN_NAME )
        IN ( SELECT CP.TABLE_CATALOG, CP.TABLE_SCHEMA, CP.TABLE_NAME,
                    CP.COLUMN_NAME
               FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES AS CP
              WHERE ( CP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                      OR
                      CP.GRANTEE IN ( SELECT ROLE_NAME
                                        FROM ENABLED_ROLES ) ) )
        AND
        C.TABLE_CATALOG = ( SELECT CATALOG_NAME
                              FROM INFORMATION_SCHEMA_CATALOG_NAME )
  ;
```

20.19 CONSTRAINT_COLUMN_USAGE view

1. Rationale: Editorial - typographical errors.

In the Definition, replace the <from clause> with:

```
  FROM ( ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
                  CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
             FROM DEFINITION_SCHEMA.CHECK_COLUMN_USAGE )
           UNION
           ( SELECT PK.TABLE_CATALOG, PK.TABLE_SCHEMA, PK.TABLE_NAME,
                    PK.COLUMN_NAME,
                    FK.CONSTRAINT_CATALOG, FK.CONSTRAINT_SCHEMA, FK.CONSTRAINT_NAME
               FROM DEFINITION_SCHEMA.REFERENTIAL_CONSTRAINTS AS FK
                    JOIN DEFINITION_SCHEMA.KEY_COLUMN_USAGE AS PK
                      ON ( FK.UNIQUE_CONSTRAINT_CATALOG,
                           FK.UNIQUE_CONSTRAINT_SCHEMA,
                           FK.UNIQUE_CONSTRAINT_NAME )
                         = ( PK.CONSTRAINT_CATALOG, PK.CONSTRAINT_SCHEMA,
                             PK.CONSTRAINT_NAME ) )
           UNION
           ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
                    CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
```

```
            FROM DEFINITION_SCHEMA.KEY_COLUMN_USAGE
                 NATURAL JOIN      DEFINITION_SCHEMA.TABLE_CONSTRAINTS
           WHERE CONSTRAINT_TYPE IN ( 'UNIQUE', 'PRIMARY KEY' ) ) )
     JOIN DEFINITION_SCHEMA.SCHEMATA
            ON ( ( TABLE_CATALOG, TABLE_SCHEMA )
                 = ( CATALOG_NAME, SCHEMA_NAME ) )
```

20.21 DATA_TYPE_PRIVILEGES view

1. *Rationale: Editorial - typographical error and missing grant statement. Do not select irrelevant rows.*

Replace the Definition with:

```
CREATE VIEW DATA_TYPE_PRIVILEGES
( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
  OBJECT_TYPE, DTD_IDENTIFIER )
AS
SELECT USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
       USER_DEFINED_TYPE_NAME, 'USER-DEFINED TYPE', DTD_IDENTIFIER
  FROM ATTRIBUTES
UNION
SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
       'TABLE', DTD_IDENTIFIER
  FROM COLUMNS
UNION
SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
       'DOMAIN', DTD_IDENTIFIER
  FROM DOMAINS
UNION
SELECT USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
       USER_DEFINED_TYPE_NAME, 'USER-DEFINED TYPE', DTD_IDENTIFIER
  FROM METHOD_SPECIFICATIONS
UNION
SELECT USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
       USER_DEFINED_TYPE_NAME, 'USER-DEFINED TYPE', DTD_IDENTIFIER
  FROM METHOD_SPECIFICATION_PARAMETERS
UNION
SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
       'ROUTINE', DTD_IDENTIFIER
  FROM PARAMETERS
UNION
SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
       'ROUTINE', DTD_IDENTIFIER
  FROM ROUTINES
 WHERE DTD_IDENTIFIER IS NOT NULL
UNION
SELECT USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
       USER_DEFINED_TYPE_NAME, 'USER-DEFINED TYPE', SOURCE_DTD_IDENTIFIER
  FROM USER_DEFINED_TYPES
 WHERE SOURCE_DTD_IDENTIFIER IS NOT NULL
UNION
SELECT USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
       USER_DEFINED_TYPE_NAME, 'USER-DEFINED TYPE', REF_DTD_IDENTIFIER
  FROM USER_DEFINED_TYPES
 WHERE REF_DTD_IDENTIFIER IS NOT NULL;

GRANT SELECT ON TABLE DATA_TYPE_PRIVILEGES
  TO PUBLIC WITH GRANT OPTION;
```

20.23 DIRECT_SUPERTYPES view

1. Rationale: Add JOIN to DEFINITION_SCHEMA.SCHEMTA table to correct view definition.

In the Definition, replace the <where clause> with:

```
WHERE ( USER_DEFINED_TYPE_CATALOG, USER_DEFINED_TYPE_SCHEMA,
        USER_DEFINED_TYPE_NAME )
      IN ( SELECT UDTP.USER_DEFINED_TYPE_CATALOG,
                  UDTP.USER_DEFINED_TYPE_SCHEMA, UDTP.USER_DEFINED_TYPE_NAME
             FROM DEFINITION_SCHEMA.USER_DEFINED_TYPE_PRIVILEGES AS UDTP
                  JOIN DEFINITION_SCHEMA.SCHEMATA AS S
                    ON ( ( UDTP.USER_DEFINED_TYPE_CATALOG,
                           UDTP.USER_DEFINED_TYPE_SCHEMA )
                         = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
             WHERE ( S.SCHEMA_OWNER = CURRENT_USER
                     OR
                     S.SCHEMA_OWNER IN ( SELECT ROLE_NAME
                                           FROM ENABLED_ROLES ) ) )
      AND
      USER_DEFINED_TYPE_CATALOG = ( SELECT CATALOG_NAME
                                      FROM INFORMATION_SCHEMA_CATALOG_NAME );
```

20.25 DOMAIN_UDT_USAGE view

1. Rationale: Domains cannot be dependent on UDTs.

Delete this entire Subclause.

20.26 DOMAINS view

1. Rationale: Domains cannot be dependent on UDTs. Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.

Replace the Definition with:

```
CREATE VIEW DOMAINS AS
  SELECT DISTINCT
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
         DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         COLLATION_CATALOG, COLLATION_SCHEMA,COLLATION_NAME,
         NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
         DATETIME_PRECISION, INTERVAL_TYPE, INTERVAL_PRECISION, DOMAIN_DEFAULT,
         MAXIMUM_CARDINALITY, D1.DTD_IDENTIFIER
    FROM DEFINITION_SCHEMA.DOMAINS AS D1
         JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D2
                LEFT JOIN DEFINITION_SCHEMA.COLLATIONS AS S
                  USING ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
              ) ON ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
                       'DOMAIN', D1.DTD_IDENTIFIER )
                     = ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
                         OBJECT_TYPE, D2.DTD_IDENTIFIER ) )
   WHERE ( ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, 'DOMAIN' )
             IN ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
```

```
                        OBJECT_TYPE
                    FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES AS UP
                   WHERE ( UP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                           OR
                           UP.GRANTEE IN ( SELECT ROLE_NAME
                                           FROM ENABLED_ROLES )
                         )
                )
            )
            OR
            ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME )
            IN ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
                   FROM COLUMNS )
            )
          AND
          ( DOMAIN_CATALOG = ( SELECT CATALOG_NAME
                               FROM INFORMATION_SCHEMA_CATALOG_NAME )
          )
;

GRANT SELECT ON TABLE DOMAINS
  TO PUBLIC WITH GRANT OPTION;
```

20.27 ELEMENT_TYPES view

1. Rationale: Remove irrelevant column from the definition of the ELEMENT_TYPES view.

In the Definition, replace the view definition with:

```
CREATE VIEW ELEMENT_TYPES AS
  SELECT DISTINCT
         E.OBJECT_CATALOG, E.OBJECT_SCHEMA, E.OBJECT_NAME,
         E.OBJECT_TYPE, ARRAY_TYPE_IDENTIFIER, DATA_TYPE,
         CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME,
         NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
         DATETIME_PRECISION, INTERVAL_TYPE, INTERVAL_PRECISION,
         USER_DEFINED_TYPE_CATALOG AS UDT_CATALOG,
         USER_DEFINED_TYPE_SCHEMA AS UDT_SCHEMA,
         USER_DEFINED_TYPE_NAME AS UDT_NAME,
         SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME,
         MAXIMUM_CARDINALITY, E.DTD_IDENTIFIER
    FROM DEFINITION_SCHEMA.ELEMENT_TYPES AS E
         JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D
                LEFT JOIN DEFINITION_SCHEMA.COLLATIONS AS S
                  USING ( COLLATION_CATALOG, COLLATION_SCHEMA,
                          COLLATION_NAME ) )
           ON ( ( E.OBJECT_CATALOG, E.OBJECT_SCHEMA, E.OBJECT_NAME,
                  E.OBJECT_TYPE, E.DTD_IDENTIFIER )
                = ( D.OBJECT_CATALOG, D.OBJECT_SCHEMA, D.OBJECT_NAME,
                    D.OBJECT_TYPE, D.DTD_IDENTIFIER ) )
   WHERE ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
           OBJECT_TYPE, ROOT_DTD_IDENTIFIER )
           IN ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
                       OBJECT_TYPE, DTD_IDENTIFIER
                  FROM INFORMATION_SCHEMA.DATA_TYPE_PRIVILEGES );
```

20.29 FIELDS view

1. Rationale: Remove irrelevant columns from the definition of the FIELDS view.

In the Definition, replace the view definition with:

```
CREATE VIEW FIELDS AS
SELECT DISTINCT
       F.OBJECT_CATALOG, F.OBJECT_SCHEMA, F.OBJECT_NAME,
       F.OBJECT_TYPE, ROW_IDENTIFIER, FIELD_NAME,
       ORDINAL_POSITION, DATA_TYPE,
       CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
       CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
       COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME,
       NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
       DATETIME_PRECISION, INTERVAL_TYPE, INTERVAL_PRECISION,
       USER_DEFINED_TYPE_CATALOG AS UDT_CATALOG,
       USER_DEFINED_TYPE_SCHEMA AS UDT_SCHEMA,
       USER_DEFINED_TYPE_NAME AS UDT_NAME,
       SCOPE_CATALOG, SCOPE_SCHEMA, SCOPE_NAME,
       MAXIMUM_CARDINALITY, F.DTD_IDENTIFIER
  FROM DEFINITION_SCHEMA.FIELDS AS F
       JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D
              LEFT JOIN DEFINITION_SCHEMA.COLLATIONS AS S
                USING ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) )
         ON ( ( F.OBJECT_CATALOG, F.OBJECT_SCHEMA, F.OBJECT_NAME,
                F.OBJECT_TYPE, F.DTD_IDENTIFIER )
                = ( D.OBJECT_CATALOG, D.OBJECT_SCHEMA, D.OBJECT_NAME,
                    D.OBJECT_TYPE, D.DTD_IDENTIFIER ) )
 WHERE ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
         OBJECT_TYPE, ROOT_DTD_IDENTIFIER )
         IN ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
                     OBJECT_TYPE, DTD_IDENTIFIER
                FROM INFORMATION_SCHEMA.DATA_TYPE_PRIVILEGES );
```

20.30 KEY_COLUMN_USAGE view

1. Rationale: Make the views deliver the information that CLI implementations actually expect.

Replace the Function with:

Identify the columns defined in this catalog that are constrained as keys and that are accessible by a given user or role.

Replace the Definition with:

```
CREATE VIEW KEY_COLUMN_USAGE AS
  SELECT KCU1.CONSTRAINT_CATALOG, KCU1.CONSTRAINT_SCHEMA,
         KCU1.CONSTRAINT_NAME, KCU1.TABLE_CATALOG,
         KCU1.TABLE_SCHEMA, KCU1.TABLE_NAME, KCU1.COLUMN_NAME,
         KCU1.ORDINAL_POSITION
    FROM DEFINITION_SCHEMA.KEY_COLUMN_USAGE AS KCU1
         JOIN INFORMATION_SCHEMA.TABLE_CONSTRAINTS AS TC
           ON ( TC.CONSTRAINT_CATALOG = KCU1.CONSTRAINT_CATALOG
                AND
                TC.CONSTRAINT_SCHEMA = KCU1.CONSTRAINT_SCHEMA
                AND
                TC.CONSTRAINT_NAME = KCU1.CONSTRAINT_NAME )
```

```
    WHERE ( ( SELECT MAX ( KCU3.ORDINAL_POSITION )
                FROM KEY_COLUMN_USAGE AS KCU3
               WHERE KCU3.CONSTRAINT_CATALOG = KCU1.CONSTRAINT_CATALOG
                     AND
                     KCU3.CONSTRAINT_SCHEMA = KCU1.CONSTRAINT_SCHEMA
                     AND
                     KCU3.CONSTRAINT_NAME = KCU1.CONSTRAINT_NAME
            )
            = ( SELECT COUNT (*)
                  FROM KEY_COLUMN_USAGE AS KCU2
                 WHERE ( KCU2.TABLE_CATALOG, KCU2.TABLE_SCHEMA,
                         KCU2.TABLE_NAME, KCU2.COLUMN_NAME )
                       IN ( SELECT CP2.TABLE_CATALOG, CP2.TABLE_SCHEMA,
                                   CP2.TABLE_NAME, CP2.COLUMN_NAME
                              FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES AS CP2
                             WHERE ( CP2.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                                     OR
                                     CP2.GRANTEE IN ( SELECT ROLE_NAME
                                                        FROM ENABLED_ROLES )
                                   )
                          )
                       AND
                       KCU2.CONSTRAINT_CATALOG =KCU1.CONSTRAINT_CATALOG
                       AND
                       KCU2.CONSTRAINT_SCHEMA = KCU1.CONSTRAINT_SCHEMA
                       AND
                       KCU2.CONSTRAINT_NAME = KCU1.CONSTRAINT_NAME
              )
            AND
            CONSTRAINT_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME )
;

GRANT SELECT ON TABLE KEY_COLUMN_USAGE
  TO PUBLIC WITH GRANT OPTION;
```

20.31 METHOD_SPECIFICATION_PARAMETERS view

1. *Rationale: Correct grant statement. Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.*

In the Definition, replace the <where clause> with:

```
WHERE ( M.USER_DEFINED_TYPE_CATALOG, M.USER_DEFINED_TYPE_SCHEMA,
        M.USER_DEFINED_TYPE_NAME )
      IN ( SELECT UDTP.USER_DEFINED_TYPE_CATALOG,
                  UDTP.USER_DEFINED_TYPE_SCHEMA,
                  UDTP.USER_DEFINED_TYPE_NAME
             FROM DEFINITION_SCHEMA.USER_DEFINED_TYPE_PRIVILEGES AS UDTP
            WHERE ( UDTP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                    OR
                    UDTP.GRANTEE IN ( SELECT ROLE_NAME
                                        FROM ENABLED_ROLES ) ) )
      AND
      M.USER_DEFINED_TYPE_CATALOG = ( SELECT CATALOG_NAME
                                        FROM INFORMATION_SCHEMA_CATALOG_NAME )
;
```

In the Definition, replace the <grant statement> with:

```
GRANT SELECT ON TABLE METHOD_SPECIFICATION_PARAMETERS
  TO PUBLIC WITH GRANT OPTION;
```

20.32 METHOD_SPECIFICATIONS view

1. *Rationale: Correct the function.*

Replace the Function with:

Identify the SQL-invoked methods in the catalog that are accessible to a given user.

2. *Rationale: Correct namespace problems associated with methods used to initialize newly-constructed structured type values. Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.*

In the Definition, replace the view definition with:

```
CREATE VIEW METHOD_SPECIFICATIONS AS
  SELECT M.SPECIFIC_CATALOG, M.SPECIFIC_SCHEMA, M.SPECIFIC_NAME,
         M.USER_DEFINED_TYPE_CATALOG AS UDT_CATALOG,
         M.USER_DEFINED_TYPE_SCHEMA AS UDT_SCHEMA,
         M.USER_DEFINED_TYPE_NAME AS UDT_NAME,
         M.METHOD_NAME, IS_STATIC, IS_OVERRIDING,
         IS_CONSTRUCTOR,
         D.DATA_TYPE, D.CHARACTER_MAXIMUM_LENGTH, D.CHARACTER_OCTET_LENGTH,
         C.CHARACTER_SET_CATALOG, C.CHARACTER_SET_SCHEMA,
         C.CHARACTER_SET_NAME,
         D.COLLATION_CATALOG, D.COLLATION_SCHEMA, D.COLLATION_NAME,
         D.NUMERIC_PRECISION, D.NUMERIC_PRECISION_RADIX, D.NUMERIC_SCALE,
         D.DATETIME_PRECISION, D.INTERVAL_TYPE, D.INTERVAL_PRECISION,
         D.USER_DEFINED_TYPE_CATALOG AS RETURN_UDT_CATALOG,
         D.USER_DEFINED_TYPE_SCHEMA AS RETURN_UDT_SCHEMA,
         D.USER_DEFINED_TYPE_NAME AS RETURN_UDT_NAME,
         D.SCOPE_CATALOG, D.SCOPE_SCHEMA, D.SCOPE_NAME,
         D.MAXIMUM_CARDINALITY, D.DTD_IDENTIFIER, M.METHOD_LANGUAGE,
         M.PARAMETER_STYLE, M.IS_DETERMINISTIC, M.SQL_DATA_ACCESS,
         M.IS_NULL_CALL,
         M.TO_SQL_SPECIFIC_CATALOG, M.TO_SQL_SPECIFIC_SCHEMA,
         M.TO_SQL_SPECIFIC_NAME,
         M.CREATED, M.LAST_ALTERED
    FROM ( DEFINITION_SCHEMA.METHOD_SPECIFICATIONS M
           JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D
                  LEFT JOIN DEFINITION_SCHEMA.COLLATIONS AS C
                    ON ( C.COLLATION_CATALOG, C.COLLATION_SCHEMA,
                         C.COLLATION_NAME )
                         = ( D.COLLATION_CATALOG, D.COLLATION_SCHEMA,
                             D.COLLATION_NAME ) )
             ON ( M.USER_DEFINED_TYPE_CATALOG, M.USER_DEFINED_TYPE_SCHEMA,
                  M.USER_DEFINED_TYPE_NAME, 'USER-DEFINED TYPE',
                  M.DTD_IDENTIFIER )
                  = ( D.OBJECT_CATALOG, D.OBJECT_SCHEMA, D.OBJECT_NAME,
                      D.OBJECT_TYPE, D.DTD_IDENTIFIER ) )
    WHERE ( M.USER_DEFINED_TYPE_CATALOG, M.USER_DEFINED_TYPE_SCHEMA,
            M.USER_DEFINED_TYPE_NAME )
          IN ( SELECT UDTP.USER_DEFINED_TYPE_CATALOG,
```

```
                    UDTP.USER_DEFINED_TYPE_SCHEMA,
                    UDTP.USER_DEFINED_TYPE_NAME
                FROM DEFINITION_SCHEMA.USER_DEFINED_TYPE_PRIVILEGES AS UDTP
             WHERE ( UDTP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                     OR
                     UDTP.GRANTEE IN ( SELECT ROLE_NAME
                                         FROM ENABLED_ROLES ) ) )
          AND
          M.USER_DEFINED_TYPE_CATALOG
          = ( SELECT CATALOG_NAME
                FROM INFORMATION_SCHEMA_CATALOG_NAME );
```

20.33 PARAMETERS view

1. *Rationale: Add missing column definitions and correct typographical error. Add JOIN to DEFINITION_SCHEMA.SCHEMTA table to correct view definition. Remove an unintentional incompatibility between SQL-92 and SQL-99, correct references to CURRENT_ROLE, recognise roles in the WHERE clause of the TABLES view and make the value of the ATTRIBUTE/COLUMN DEFAULT visible to any user who has some privilege on the attribute/column.*

Replace the Definition with:

```
CREATE VIEW PARAMETERS AS
  SELECT P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME,
         P1.ORDINAL_POSITION, P1.PARAMETER_MODE, P1.IS_RESULT,
         P1.AS_LOCATOR, P1.PARAMETER_NAME, P1.FROM_SQL_SPECIFIC_CATALOG,
         P1.FROM_SQL_SPECIFIC_SCHEMA, P1.FROM_SQL_SPECIFIC_NAME,
         P1.TO_SQL_SPECIFIC_CATALOG, P1.TO_SQL_SPECIFIC_SCHEMA,
         P1.TO_SQL_SPECIFIC_NAME, D1.DATA_TYPE,
         D1.CHARACTER_MAXIMUM_LENGTH, D1.CHARACTER_OCTET_LENGTH,
         C1.CHARACTER_SET_CATALOG,
         C1.CHARACTER_SET_SCHEMA, C1.CHARACTER_SET_NAME, C1.COLLATION_CATALOG,
         C1.COLLATION_SCHEMA, C1.COLLATION_NAME, D1.NUMERIC_PRECISION,
         D1.NUMERIC_PRECISION_RADIX, D1.NUMERIC_SCALE, D1.DATETIME_PRECISION,
         D1.INTERVAL_TYPE, D1.INTERVAL_PRECISION, D1.UDT_CATALOG,
         D1.UDT_SCHEMA, D1.UDT_NAME, D1.SCOPE_CATALOG,
         D1.SCOPE_SCHEMA, D1.SCOPE_NAME, D1.MAXIMUM_CARDINALITY,
         D1.DTD_IDENTIFIER
    FROM ( DEFINITION_SCHEMA.PARAMETERS P1
           LEFT JOIN ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D1
                       LEFT JOIN DEFINITION_SCHEMA.COLLATIONS C1
                         ON ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA,
                                C1.COLLATION_NAME )
                              = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA,
                                  D1.COLLATION_NAME ) ) )
             ON ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
                  'ROUTINE', P1.DTD_IDENTIFIER )
                = ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
                    OBJECT_TYPE, D1.DTD_IDENTIFIER ) )
           JOIN DEFINITION_SCHEMA.ROUTINES R1
             ON ( ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME )
                  = ( R1.SPECIFIC_CATALOG, R1.SPECIFIC_SCHEMA,
                      R1.SPECIFIC_NAME )
         )
    WHERE ( ( ( MODULE_CATALOG, MODULE_SCHEMA, MODULE_NAME ) IS NULL
              AND
              ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA, P1.SPECIFIC_NAME )
              IN ( SELECT SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME
                     FROM DEFINITION_SCHEMA.ROUTINE_PRIVILEGES
                    WHERE ( GRANTEE IN ( 'PUBLIC', CURRENT_USER )
```