



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)

© ISO/IEC 2003 – All rights reserved

Published in Switzerland

AVA Documents
Document Sharing Hub

Contents

	Page
2 Normative references	9
3.1.1 Definitions taken from ISO/IEC 10646	9
3.1.2 Definitions taken from Unicode	9
3.1.5 Definitions provided in Part 2	10
3.3.1.2 Other terms	12
4.1 Data types	12
4.2 Character strings	15
4.2.1 Character strings and collating sequences	16
4.2.2.1 Operators that operate on character strings and return character strings	17
4.2.4 Named character sets	17
4.3 Binary strings	21
4.3.1 Binary string comparison	22
4.4.1 Bit string comparison and assignment	22
4.5 Numbers	22
4.5.1 Characteristics of numbers	23
4.5.2 Operations involving numbers	24
4.6.1 Comparison and assignment of booleans	24
4.7.1 Datetimes	24
4.7.2 Intervals	25
4.7.3 Operations involving datetimes and intervals	25
4.8 User-defined types	26
4.8.1 Observers and mutators	27
4.8.2 Constructors	27
4.8.4 User-defined type comparison and assignment	28
4.9 Row types	28
4.10 Reference types	28

4.11 Collection types	29
4.11.2 Collection comparison	30
4.12 Type conversions and mixing of data types	30
4.13 Data conversions	30
4.16 Tables	30
4.16.1 Types of tables	31
4.16.3 Operations involving tables	32
4.17 Integrity constraints	32
4.17.1 Checking of constraints	33
4.17.2 Table constraints	33
4.18.1 General rules and definitions	34
4.18.4 Known functional dependencies in a <joined table>	34
4.18.9 Known functional dependencies in the result of <having clause>	34
4.18.10 Known functional dependencies in a <query specification>	35
4.20 SQL-schemas	35
4.21 SQL-client modules	35
4.23 SQL-invoked routines	36
4.24 Built-in functions	39
4.25 SQL-paths	40
4.26.4 Locators	40
4.27 Diagnostics area	40
4.30.1 Classes of SQL-statements	41
4.30.2 SQL-statements classified by functions	42
4.30.3 SQL-statements and transaction states	42
4.30.4 SQL-statement atomicity	43
4.31.1 Authorization identifiers	44
4.32 SQL-transactions	44
4.34 SQL-sessions	45
4.34.1 Execution contexts	45
4.35 Triggers	46
4.35.2 Execution of triggers	46
4.36 Client-server operation	47
5.2 <token> and <separator>	47
5.3 <literal>	51
5.4 Names and identifiers	52
6.1 <data type>	53
6.3 <value specification> and <target specification>	56
6.5 <identifier chain>	58
6.6 <column reference>	60
6.8 <field reference>	62
6.11 <method invocation>	62
6.14 <dereference operation>	63
6.16 <set function specification>	63
6.17 <numeric value function>	65
6.18 <string value function>	66
6.19 <datetime value function>	72
6.20 <interval value function>	72
6.22 <cast specification>	73
6.23 <value expression>	77
6.24 <new specification>	79
6.25 <subtype treatment>	80
6.26 <numeric value expression>	81

6.27 <string value expression>	81
6.28 <datetime value expression>	83
6.29 <interval value expression>	84
6.30 <boolean value expression>	85
7.1 <row value constructor>	85
7.2 <row value expression>	86
7.3 <table value constructor>	87
7.6 <table reference>	87
7.7 <joined table>	93
7.8 <where clause>	94
7.9 <group by clause>	94
7.10 <having clause>	102
7.11 <query specification>	102
7.12 <query expression>	105
7.13 <search or cycle clause>	110
7.14 <subquery>	113
8.2 <comparison predicate>	113
8.3 <between predicate>	115
8.4 <in predicate>	115
8.5 <like predicate>	116
8.6 <similar predicate>	117
8.7 <null predicate>	119
8.8 <quantified comparison predicate>	119
8.10 <unique predicate>	120
8.11 <match predicate>	120
8.12 <overlaps predicate>	122
8.13 <distinct predicate>	123
8.14 <type predicate>	123
9.0 Determination of identical values	123
9.1 Retrieval assignment	125
9.2 Store assignment	126
9.3 Data types of results of aggregations	126
9.5 Type precedence list	127
10.3 <path specification>	127
10.4 <routine invocation>	127
10.5 <privileges>	136
10.6 <character set specification>	137
10.7 <specific routine designator>	138
10.8 <collate clause>	141
10.12 Execution of triggers	141
10.13 Execution of array-returning functions	141
10.14 Data type identity	143
11.1 <schema definition>	144
11.2 <drop schema statement>	145
11.3 <table definition>	146
11.4 <column definition>	149
11.5 <default clause>	150
11.7 <unique constraint definition>	150
11.8 <referential constraint definition>	151
11.9 <check constraint definition>	157
11.16 <drop column scope clause>	157
11.17 <drop column definition>	158

11.19 <drop table constraint definition>	158
11.20 <drop table statement>	159
11.21 <view definition>	159
11.22 <drop view statement>	161
11.23 <domain definition>	161
11.30 <character set definition>	162
11.31 <drop character set statement>	162
11.32 <collation definition>	163
11.33 <drop collation statement>	163
11.35 <translation definition>	163
11.36 <assertion definition>	164
11.37 <drop assertion statement>	164
11.38 <trigger definition>	166
11.40 <user-defined type definition>	166
11.41 <attribute definition>	177
11.43 <add attribute definition>	178
11.44 <drop attribute definition>	178
11.45 <add original method specification>	179
11.46 <add overriding method specification>	183
11.47 <drop method specification>	187
11.48 <drop data type statement>	190
11.49 <SQL-invoked routine>	191
11.50 <alter routine statement>	200
11.51 <drop routine statement>	201
11.52 <user-defined cast definition>	201
11.53 <drop user-defined cast statement>	203
11.54 <user-defined ordering definition>	203
11.55 <drop user-defined ordering statement>	205
11.56 <transform definition>	207
11.57 <drop transform statement>	207
12.2 <grant privilege statement>	208
12.4 <select statement: single row>	208
12.6 <revoke statement>	209
13.1 <SQL-client module definition>	211
13.2 <module name clause>	211
13.3 <externally-invoked procedure>	212
13.4 Calls to an <externally-invoked procedure>	212
13.5 <SQL procedure statement>	214
13.6 Data type correspondences	217
14.1 <declare cursor>	217
14.3 <fetch statement>	220
14.4 <close statement>	221
14.5 <select statement: single row>	221
14.6 <delete statement: positioned>	222
14.8 <insert statement>	223
14.9 <update statement: positioned>	224
14.10 <update statement: searched>	226
14.11 <temporary table declaration>	227
14.12 <free locator statement>	227
14.14 Effect of deleting rows from base table	227
14.17 Effect of inserting tables into base tables	228
14.18 Effect of inserting a table into a derived table	229

14.20 Effect of replacing rows from base table	229
15.2 <return statement>	229
16.4 <savepoint statement>	230
16.5 <release savepoint statement>	230
16.6 <commit statement>	230
16.7 <rollback statement>	230
19.1 <get diagnostics statement>	231
19.2 Pushing and popping the diagnostics area stack	234
20.4 CARDINAL_NUMBER domain	235
20.9 APPLICABLE_ROLES view	235
20.11 ATTRIBUTES view	235
20.12 CHARACTER_SETS view	236
20.12a CHECK_CONSTRAINT_ROUTINE_USAGE view	237
20.14 COLLATION view	237
20.16 COLUMN_PRIVILEGES view	238
20.17 COLUMN_UDT_USAGE view	238
20.18 COLUMNS view	239
20.19 CONSTRAINT_COLUMN_USAGE view	240
20.21 DATA_TYPE_PRIVILEGES view	241
20.23 DIRECT_SUPERTYPES view	242
20.25 DOMAIN_UDT_USAGE view	242
20.26 DOMAINS view	242
20.27 ELEMENT_TYPES view	243
20.29 FIELDS view	244
20.30 KEY_COLUMN_USAGE view	244
20.31 METHOD_SPECIFICATION_PARAMETERS view	245
20.32 METHOD_SPECIFICATIONS view	246
20.33 PARAMETERS view	247
20.34 REFERENCED_TYPES view	248
20.35 REFERENTIAL_CONSTRAINTS view	249
20.38 ROLE_TABLE_GRANTS view	249
20.43 ROUTINE_PRIVILEGES view	250
20.44a ROUTINE_ROUTINE_USAGE view	250
20.45 ROUTINES view	251
20.53 TABLE_CONSTRAINTS view	252
20.54 TABLE_METHOD_PRIVILEGES view	252
20.55 TABLE_PRIVILEGES view	253
20.56 TABLES view	253
20.57 TRANSFORMS view	254
20.58 TRANSLATIONS view	254
20.59 TRIGGERED_UPDATE_COLUMNS view	255
20.60a TRIGGER_ROUTINE_USAGE view	255
20.62 TRIGGERS view	256
20.63 USAGE_PRIVILEGES view	258
20.64 UDT_PRIVILEGES view	258
20.65 USER_DEFINED_TYPES view	258
20.66a VIEW_ROUTINE_USAGE view	259
20.62 TRIGGERS view	260
20.68 VIEWS view	260
20.69 Short name views	261
20.70 Definition of SQL built-in functions	266
21.3 EQUAL_KEY_DEGREES assertion	266

21.6 ASSERTIONS base table	267
21.7 ATTRIBUTES base table	267
20.7a CHARACTER_ENCODING_FORMS base table	267
21.7b CHARACTER_REPERTOIRES base table	268
21.8 CHARACTER_SETS base table	269
21.9a CHECK_CONSTRAINT_ROUTINE_USAGE base table	270
21.12 COLLATIONS base table	271
21.14 COLUMNS base table	272
21.15 DATA_TYPE_DESCRIPTOR base table	272
21.16 DIRECT_SUPERTABLES	277
21.23 METHOD_SPECIFICATION_PARAMETERS	277
21.24 METHOD_SPECIFICATIONS base table	278
21.25 PARAMETERS base table	279
21.30 ROUTINE_COLUMN_USAGE base table	279
21.31 ROUTINE_PRIVILEGES base table	279
21.31a ROUTINE_ROUTINE_USAGE base table	279
21.32 ROUTINE_TABLE_USAGE base table	280
21.33 ROUTINES base table	280
21.34 SCHEMATA base table	282
21.35 SQL_FEATURES base table	282
21.36 SQL_IMPLEMENTATION_INFO base table	283
21.37 SQL_LANGUAGES base table	284
21.38 SQL_SIZING base table	284
21.39 SQL_SIZING_PROFILES base table	285
21.43 TABLES base table	285
21.44 TRANSFORMS base table	285
21.45 TRANSLATIONS base table	286
21.47 TRIGGER_COLUMN_USAGE base table	286
21.47a TRIGGER_ROUTINE_USAGE base table	287
21.48 TRIGGER_TABLE_USAGE base table	288
21.49 TRIGGERS base table	288
21.52 USER_DEFINED_TYPES base table	289
21.54a VIEW_ROUTINE_USAGE base table	290
21.56 VIEWS base table	291
22.1 SQLSTATE	291
23.1 General conformance requirements	293
Annex A SQL conformance summary	293
Annex B Implementation-defined elements	299
Annex C Implementation-dependent elements	303
Annex E Incompatibilities with ISO/IEC 9075:1992 and ISO/IEC 9075-4:1996	304
Annex F SQL feature and package taxonomy	307

(Blank page)

Withdrawn

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 Form at 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 $n \cdot 10^S$, where n is an integer such that $-(R^P) \leq n < RP$.
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 $M \cdot 10^E$ 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”:

<i>SV</i>	Source value
<i>TV</i>	Target value
<i>.UTC</i>	UTC component of <i>SV</i> or <i>TV</i> (if source or target has time zone)
<i>.TZ</i>	Time zone component of <i>SV</i> or <i>TV</i> (if source or target has time zone)
<i>STZD</i>	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 mutators.*

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 AV and for every attribute $A'(A' \neq A)$ 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 " $T: A \twoheadrightarrow B$ " (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 " $A \twoheadrightarrow B$ " 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) $\leq i \leq n$, 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 $\{CRA_1, \dots, CRA_N\} \twoheadrightarrow \{CRBC\}$ in case one comparand is a deterministic expression of column references $CRA_1, \dots, CRA_N$ 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 $OP1, OP2, \dots$ 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 $\{OP1, OP2, \dots\} \rightarrow CC$ is a known functional dependency in TI .

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

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 characteristic>s 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>
- <savpoint 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.