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

Published 2003-06-01

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

Information technology — Database languages — SQL —

Part 2:
Foundation (SQL/Foundation)

TECHNICAL CORRIGENDUM 2

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

Partie 2: Fondations (SQL/Fondations)

RECTIFICATIF TECHNIQUE 2

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

Statement of purpose for rationale:

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

Notes on numbering:

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

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

Published in Switzerland

Contents

Page

(Blank page)

Information technology — Database languages — SQL —

Part 2:
Foundation (SQL/Foundation)

TECHNICAL CORRIGENDUM 2

2 Normative references

1. Rationale: Add missing references..

Insert the following items to this Subclause:

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

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

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

3.1.1 Definitions taken from ISO/IEC 10646

1. Rationale: Remove unused definitions.

Replace the Subclause with:

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

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

b) repertoire

3.1.2 Definitions taken from Unicode

1. Rationale: Remove unused definitions.

Replace the Subclause with:

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

a) character encoding form

b) code point

c) control character

d) transcoding

3.1.5 Definitions provided in Part 2

1. Rationale: Clarify assignable and comparable.

Replace item a) with:

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

2. Rationale: Clarify the definition of "assignment"

Replace item b) with:

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

3. Rationale: Clarify the definition of "comparable"

Replace item i) with:

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

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

Replace item l) with:

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

Two null values of comparable type are not distinct.

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

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

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

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

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

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

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

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

5. *Rationale: Define "equals".*

Insert the following definition:

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

6. *Rationale: Define "identical"*

Insert the following definition:

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

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

Replace item rr) with:

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

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

In item vv) delete the following bullet items:

— U+200C, Zero Width Non-Joiner

— U+200D, Zero Width Joiner

— U+200E, Left-To-Right Mark

— U+200F, Right-To-Left Mark

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

In item vv) insert the following bullet item:

— U+202F, Narrow No-Break Space

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

In item vv), delete the following bullet item:

— U+200B, Zero Width Space

3.3.1.2 Other terms

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

Insert the following Subclause:

3.3.1.2 Other terms

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

4.1 Data types

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

Replace the 3rd paragraph with:

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

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

Replace the 11th paragraph with:

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

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

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

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

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

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

Insert the following paragraph immediately after the 15th paragraph:

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

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

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

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

— *TY* is *UDT*;

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

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

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

5. Rationale: Provide consistent rules for comparison operations.

Insert the following paragraphs after the last paragraph:

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

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

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

1. *T* is a member of *S*.

2. *T* is a row type and the declared type of some field of *T* is *S*-ordered.

3. *T* is an array type and the element type of *T* is *S*-ordered.

4. *T* is a structured type whose comparison form is STATE and the declared type of some attribute of *T* is *S*-ordered.

5. *T* is a user-defined type whose comparison form is MAP and the return type of the SQL-invoked function that is identified by the <map function specification> is *S*-ordered.

6. *T* is a reference type with a derived representation and the declared type of some attribute enumerated by the <derived representation> is *S*-ordered.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Insert the following paragraph after the last paragraph:

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

— *DT* is a constituent of *DT*.

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

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

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

7. *Rationale: Clarify assignable and comparable.*

Insert the following paragraph after the last paragraph:

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

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

4.2 Character strings

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

Replace the 1st and 2nd paragraphs with:

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

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

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

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

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

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

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

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

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

Character sets fall into three categories: those defined by national or international standards, those defined by SQL-implementations, and those defined by applications. The character sets defined by ISO/IEC 10646 and The Unicode Standard are known as *Universal Character Sets* (UCS) and their treatment is described in Subclause 4.2.4, "Universal character sets". Every character set contains the <space> character (equivalent to U+0020). An application defines a character set by assigning a new name to a character set from one of the first two categories. They can be defined to "reside" in any schema chosen by the application. Character sets defined by standards or by SQL-implementations reside in the Information Schema (named INFORMATION_SCHEMA) in each catalog, as do collations defined by standards and collations, transliterations, and transcodings defined by SQL-implementations.