INTERNATIONAL STANDARD ISO/IEC 9075-2:2003
TECHNICAL CORRIGENDUM 1

Published 2005-11-15

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 1

Technologies de l'information — Langages de base de données — SQL —
Partie 2: Fondations (SQL/Foundation)

RECTIFICATIF TECHNIQUE 1

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

Statement of purpose for rationale

A statement indicating the rationale for each change to ISO/IEC 9075-2:2003 is included. This is to inform the users of ISO/IEC 9075-2:2003 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 Technical 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 Technical Corrigendum introduces new subclauses, the new subclauses have been numbered as follows:

> Subclauses inserted between, for example, 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 9075-2:2003/Cor.1:2005(E)**

Published in Switzerland

Contents

Page

Tables

Table

Page

Information technology — Database languages — SQL —

Part 2:
Foundation (SQL/Foundation)

TECHNICAL CORRIGENDUM 1

Foreword

1. *Rationale: Correct intent of this second edition.*

   Replace the 6th paragraph with:

   This second edition of ISO/IEC 9075-2, together with ISO/IEC 9075-11:2003, cancels and replaces ISO/IEC 9075-2:1999 and ISO/IEC 9075-5:1999, which have been technically revised. It also incorporates the Amendments ISO/IEC 9075-2:1999/Amd.1:2001 and ISO/IEC 9075-5:1999/Amd.1:2001 and the Technical Corrigenda ISO/IEC 9075-2:1999/Cor.1:2000, ISO/IEC 9075-2:1999/Amd.1:2001/Cor.1:2003, ISO/IEC 9075-2:1999/Cor.2:2003, ISO/IEC 9075-5:1999/Cor.1:2000, ISO/IEC 9075-5:1999/Amd.1:2001/Cor.1:2003 and ISO/IEC 9075-5:1999/Cor.2:2003.

2. *Rationale: Remove incorrect reference to obsolete part.*

   In the 7th paragraph, delete the 5th bullet.

3 Definitions, notations, and conventions

3.1 Definitions

3.1.6 Definitions provided in Part 2

1. *Rationale: Editorial.*

   Replace the definition of "assignable" with:

   3.1.6.1 **assignable (of types, taken pairwise):** 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* and *T2* may be the same data type).

2. *Rationale: Restore the correct definition of "assignment".*

   Replace the definition of "assignment" with:

   3.1.6.2 **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: Remove erroneous word "comparable" from definition.*

   Replace the definition of "identical" with:

   3.1.6.15 **identical (of a pair of values):** Indistinguishable, in the sense that it is impossible, by any means specified in ISO/IEC 9075, to detect any difference between them. For the full definition, see Subclause 9.8, "Determination of identical values".

4. *Rationale: Definitions required for correction to dynamic result sets.*

   Insert the following definitions:

   3.1.6.29.1 **result set:** A sequence of rows brought into existence by opening a cursor and ranged over by that cursor.

   3.1.6.29.2 **result set sequence:** A sequence of returned result sets.

   3.1.6.29.3 **returned result set:** A result set created during execution of an SQL-invoked procedure and not destroyed when that execution terminates. Such a result set can be accessed by using a cursor other than the one that brought it into existence.

   3.1.6.42.1 **with-return cursor:** A cursor that, when opened, creates a result set that is capable of becoming a returned result set. The WITH RETURN option of <declare cursor> and <allocate cursor statement> specifies a with-return cursor.

4 Concepts

4.13 Columns, fields and attributes

1. *Rationale: Missing item in the column descriptor.*

   In the 6th paragraph, insert the following item to the list of items in a column descriptor:

   — An indication of whether *C* is updatable or not.

4.14 Tables

4.14.2 Types of tables

1. *Rationale: Editorial.*

   Replace the 1st paragraph with:

   A table is either a base table, a derived table, a transient table, or a viewed table. A base table is either a persistent base table, a global temporary table, a created local temporary table, or a declared local temporary table.

2. *Rationale: Delete obsolete reference to element type of a <query expression>.*

   Replace the 3rd paragraph with:

   A derived table is a table derived directly or indirectly from one or more other tables by the evaluation of a <query expression>.

3. *Rationale: Editorial.*

   Append the following text to the 4th paragraph:

   Base tables and views are identified by <table name>s. The same <table name>, in its fully qualified form, cannot be used for both a base table and a view.

4. *Rationale: Define view components, for use in view component privilege descriptors.*

   Insert the following paragraph after the 4th paragraph:

   A <query specification>, <table value constructor>, <explicit table>, or <query expression> contained in a <view definition> is called a *view component*.

4.14.3 Table descriptors

1. *Rationale: Track simple updatability in derived table descriptors.*

   In the 5th paragraph, insert the following item to the list of items in a derived table descriptor:

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

4.17 Integrity constraints

4.17.1 Overview of integrity constraints

1. *Rationale: Every constraint descriptor is to include a <search condition>.*

   Insert the following at the end of the 1st paragraph:

   — An applicable <search condition>.

   NOTE 28.1 — The applicable <search condition> included in the descriptor is not necessarily the <search condition> that might be contained in the SQL-statement whose execution brings the constraint descriptor into existence. The General Rules for the SQL-statement in question specify the applicable <search condition> to be included in the constraint descriptor, in some cases deriving it from a given <search condition>. For example, the syntax for table constraints allows universal quantification over the rows of the table in question to be implicit; in the applicable <search condition> included in the descriptor, that universal quantification is made explicit, to allow for uniform treatment of all types of constraint.

4.17.2 Checking of constraints

1. *Rationale: Clarify when constraints are checked.*

   Replace the entire Subclause with:

   Every constraint is either *deferrable* or *non-deferrable*. Within an SQL-transaction, every constraint has a constraint mode; if a constraint is nondeferrable, then its constraint mode is always *immediate*; otherwise, it is either immediate or *deferred*. Every constraint has an initial constraint mode that specifies the constraint mode for that constraint at the start of each SQL-transaction and immediately after definition of that constraint. If a constraint is deferrable, then its constraint mode within the current SQL-transaction may be changed (from immediate to deferred, or from deferred to immediate) by execution of a <set constraints mode statement>.

   The checking of a constraint depends on its constraint mode within the current SQL-transaction. Whenever an SQL-statement is executed, every constraint whose mode is immediate is checked, at a certain point after any changes to SQL-data and schemas resulting from that execution have been effected, to see if it is *satisfied*. A constraint is *satisfied* if and only if the applicable <search condition>s included in its descriptor evaluates to *True* or *Unknown*. If any constraint is not satisfied, then any changes to SQL-data or schemas resulting from executing that statement are canceled. (See the General Rules of Subclause 13.5, "<SQL procedure statement>".)

   NOTE 29 — This includes SQL-statements that are executed as a direct result or an indirect result of executing a different SQL-statement. It also includes statements whose effects explicitly or implicitly include setting the constraint mode to immediate.

   The constraint mode can be set to immediate either explicitly by execution of a <set constraints mode statement>, or implicitly at the end of the current SQLtransaction.

   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>.

4.17.3 Table constraints

1. *Rationale: Every constraint descriptor includes a <search condition>.*

   Replace the 1st paragraph with:

   A constraint whose definition is part of some base table definition is a *table constraint*. Being part of a particular table definition allows for convenient syntactic shorthands in which universal quantification over the rows of the table in question is implied.

   A table constraint is either a unique constraint, a referential constraint, or a table check constraint. A table constraint descriptor is either a unique constraint descriptor, a referential constraint descriptor, or a table check constraint descriptor, respectively.

2. *Rationale: Every constraint descriptor includes a <search condition>.*

   Delete the 11th and 12th paragraphs

4.17.4 Domain constraints

1. *Rationale: Every constraint descriptor includes a <search condition>, and clarify how domain constraints are applied.*

   Replace the entire Subclause with:

   A domain constraint is a constraint that is specified for a domain. It is applied to all columns that are based on that domain, and to all values cast to that domain.

   A domain constraint is described by a domain constraint descriptor. In addition to the components of every constraint descriptor, a domain constraint descriptor includes:

   — The template <search condition> for the generation of domain constraint usage <search condition>s.

   — A possibly empty set of domain constraint usages.

   A domain constraint usage descriptor is created implicitly by the evaluation of a <column definition> whose <data type or domain name> is a <domain name>. If *C* is such a column and *D* is the domain identified by the <domain name>, then every domain constraint *DC* defined for *D* implies a domain constraint usage, to the effect that each value in *C* satisfies *DC*.

   In addition to the components of every constraint descriptor, a domain constraint usage descriptor includes:

   — The name of the applicable column.

   — The applicable <search condition> that evaluates whether each value in *C* satisfies *DC*.

   A domain constraint is satisfied by SQL-data if and only if, for every table *T* that has a column named *C* based on that domain, the applicable <search condition> recorded in the appropriate domain constraint usage evaluates to *True* or *Unknown*.

A domain constraint is satisfied by the result of a <cast specification> if and only if the specified template <search condition>, with each occurrence of the <general value specification> VALUE replaced by that result, evaluates to *True* or *Unknown*.

4.17.5 Assertions

1. *Rationale: Every constraint descriptor includes a <search condition>.*

   Replace the entire Subclause with:

   An assertion is a constraint whose descriptor is an independent schema component not included in any table descriptor.

4.18 Functional dependencies

4.18.6 Known functional dependencies in a <joined table>

1. *Rationale: Rules for inheriting funtional dependencies in <joined table> were omitted.*

   Insert the following after the 5th paragraph:

   If $A \mapsto B$ is a known functional dependency in *T1*, *CA* is the counterpart of *A* in *R*, and *CB* is the counterpart of *B* in *R*, then $CA \mapsto CB$ is a known functional dependency in *R* when one of the following is true:

   — CROSS, INNER, or LEFT is specified.

   — RIGHT or FULL is specified and at least one column in *A* is known not nullable.

   If $A \mapsto B$ is a known functional dependency in *T2*, *CA* is the counterpart of *A* in *R*, and *CB* is the counterpart of *B* in *R*, then $CA \mapsto CB$ is a known functional dependency in *R* when one of the following is true:

   — CROSS, INNER, or RIGHT is specified.

   — LEFT or FULL is specified and at least one column in *A* is known not nullable.

4.18.15 Known functional dependencies in a <query expression>

1. *Rationale: Remove incorrect reference to <joined table>.*

   Replace the 2nd paragraph with:

   A <query expression> that is a <query term> that is a <query primary> that is a <simple table> is covered by previous Subclauses of this Clause.

4.20 SQL-Schemas

1. *Rationale: Editorial.*

   Replace the 5th paragraph with:

   Base tables and views are identified by <table name>s. A <table name> consists of a <schema name>, followed by a <period>, followed by an <identifier>. The <schema name> identifies the schema that includes the table descriptor of the base table or view identified by the <table name>. The <table name>s of base tables and views defined in different schemas can have equivalent <identifier>s.

   NOTE 44.1 — Equivalence of <identifier>s is defined in Subclause 5.2, "<token> and <separator>".

4.27 SQL-invoked routines

4.27.3 Execution of SQL-invoked routines

1. *Rationale: The treatment of the authorization stack is inconsistent with Subclauses 4.34.1.1 and 10.4.*

   Replace the 1st, 2nd, 3rd, 4th, 5th, 6th and 7th paragraphs with:

   When an SQL-invoked routine is invoked, a copy of the current SQL-session context is pushed onto the stack and some values are modified (see the General Rules of Subclause 10.4, "<routine invocation>") before the <routine body> is executed. The treatment of the authorization stack is described in Subclause 4.34.1.1, "SQL-session authorization identifiers".

4.27.4 Routine descriptors

1. *Rationale: Change of terminology.*

   In the 1st paragraph, replace the 8th bullet with:

   — If the SQL-invoked routine is an SQL-invoked procedure, then the maximum number of returned result sets.

2. *Rationale: An external routine does not have two authorization identifiers.*

   In the 1st paragraph, in the 15th bulleted item, delete the 5th bulleted subitem ("The external routine authorization identifier ... ").