



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

Published 2007-04-15

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 2

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

Partie 2: Fondations (SQL/Foundation)

RECTIFICATIF TECHNIQUE 2

Technical Corrigendum 2 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*. Corrigendum 2 cancels and replaces ISO/IEC 9075-2:2003/Cor.1:2005.

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.2:2007(E)

© ISO/IEC 2007 – All rights reserved

Published in Switzerland

AVA Documents
Document Sharing Hub

Contents

Page

Foreword.....	1
3 Definitions, notations, and conventions.....	1
3.1 Definitions.....	1
3.1.6 Definitions provided in Part 2.....	1
4 Concepts.....	2
4.13 Columns, fields and attributes.....	2
4.14 Tables.....	3
4.14.2 Types of tables.....	3
4.14.3 Table descriptors.....	3
4.17 Integrity constraints.....	4
4.17.1 Overview of integrity constraints.....	4
4.17.2 Checking of constraints.....	4
4.17.3 Table constraints.....	5
4.17.4 Domain constraints.....	5
4.17.5 Assertions.....	6
4.18 Functional dependencies.....	6
4.18.6 Known functional dependencies in a <joined table>.....	6
4.18.15 Known functional dependencies in a <query expression>.....	6
4.20 SQL-Schemas.....	7
4.27 SQL-invoked routines.....	7
4.27.3 Execution of SQL-invoked routines.....	7
4.27.4 Routine descriptors.....	7
4.27.5 Result sets returned by SQL-invoked procedures.....	8
4.32 Cursors.....	9
4.32.1 General description of cursors.....	9
4.32.2 Operations on and using cursors.....	9
4.33 SQL-statements.....	10
4.33.3 SQL-statements and SQL-data access indication.....	10
4.33.4 SQL-statements and transaction states.....	10
4.33.5 SQL-statement atomicity and statement execution contexts.....	12
4.34 Basic security model.....	13
4.34.1 Authorization identifiers.....	13
4.34.1.1 SQL-session authorization identifiers.....	13
4.34.2 Privileges.....	13
4.34.3 Roles.....	14

4.34.4	Security model definitions.....	15
4.37	SQL-sessions.....	16
4.37.3	SQL-session properties.....	16
4.38	Triggers.....	17
4.38.2	Trigger execution.....	17
5	Lexical elements.....	17
5.1	<SQL terminal character>.....	17
5.2	<token> and <separator>.....	17
5.3	<literal>.....	18
5.4	Names and identifiers.....	19
6	Scalar expressions.....	21
6.1	<data type>.....	21
6.4	<value specification> and <target specification>.....	21
6.6	<identifier chain>.....	23
6.7	<column reference>.....	23
6.9	<set function specification>.....	24
6.10	<window function>.....	24
6.12	<cast specification>.....	24
6.13	<next value expression>.....	30
6.27	<numeric value function>.....	30
6.29	<string value function>.....	31
7	Query expressions.....	31
7.1	<row value constructor>.....	31
7.6	<table reference>.....	31
7.7	<joined table>.....	34
7.8	<where clause>.....	36
7.9	<group by clause>.....	36
7.10	<having clause>.....	37
7.11	<window clause>.....	37
7.12	<query specification>.....	37
7.13	<query expression>.....	40
7.14	<search or cycle clause>.....	44
7.15	<subquery>.....	45
8	Predicates.....	46
8.2	<comparison predicate>.....	46
9	Additional common rules.....	47
9.24	Determination of view and view component privileges.....	47
10	Additional common elements.....	52
10.4	<routine invocation>.....	52
10.8	<constraint name definition> and <constraint characteristics>.....	54
10.9	<aggregate function>.....	55

11	Schema definition and manipulation.	55
11.2	<drop schema statement>.	55
11.3	<table definition> .	56
11.4	<column definition> .	56
11.6	<table constraint definition> .	57
11.7	<unique constraint definition> .	58
11.8	<referential constraint definition> .	58
11.9	<check constraint definition> .	64
11.18	<drop column definition> .	64
11.22	<view definition> .	65
11.24	<domain definition>.	66
11.30	<drop domain statement>.	67
11.37	<assertion definition>.	67
11.41	<user-defined type definition>.	68
11.42	<attribute definition>.	68
11.50	<SQL-invoked routine>.	69
11.51	<alter routine statement>.	69
12	Access control.	70
12.1	<grant statement>.	70
12.2	<grant privilege statement>.	70
12.3	<privileges>.	71
12.4	<role definition>.	71
12.5	<grant role statement>.	72
12.6	<drop role statement>.	73
12.7	<revoke statement>.	73
12.8	Grantor determination.	88
13	SQL-client modules.	89
13.4	Calls to an <externally-invoked procedure> .	89
13.5	<SQL procedure statement>.	90
14	Data manipulation.	92
14.1	<declare cursor> .	92
14.2	<open statement>.	93
14.3	<fetch statement>.	93
14.4	<close statement>.	93
14.6	<delete statement: positioned> .	94
14.7	<delete statement: searched> .	94
14.8	<insert statement> .	95
14.9	<merge statement> .	97
14.10	<update statement: positioned> .	100
14.11	<update statement: searched> .	101
14.16	Effect of deleting rows from base tables. .	102
14.17	Effect of deleting some rows from a derived table .	102

14.19	Effect of inserting tables into base tables.	103
14.20	Effect of inserting a table into a derived table.	104
14.21	Effect of inserting a table into a viewed table	104
14.22	Effect of replacing rows in base tables.	105
14.23	Effect of replacing some rows in a derived table	106
14.24	Effect of replacing some rows in a viewed table	106
14.27	Execution of triggers.	107
14.28	Effect of opening a cursor.	107
14.29	Determination of the current row of a cursor	109
14.30	Effect of closing a cursor.	111
16	Transaction management.	113
16.1	<start transaction statement>.	113
16.2	<set transaction statement>.	114
16.3	<set constraints mode statement>	114
16.6	<commit statement>.	115
16.7	<rollback statement>.	116
16.8	<transaction characteristics>.	117
17	Connection management.	119
17.1	<connect statement>.	119
18	Session management.	119
18.1	<set session characteristics statement>	119
18.2	<set session user identifier statement>	120
18.3	<set role statement>.	120
19	Dynamic SQL.	121
19.2	<allocate descriptor statement>	121
19.3	<deallocate descriptor statement>	122
19.4	<get descriptor statement>	122
19.5	<set descriptor statement>	122
19.6	<prepare statement>	122
19.8	<deallocate prepared statement>	123
19.9	<describe statement>	123
19.10	<input using clause>	124
19.11	<output using clause>	124
19.12	<execute statement>	124
19.13	<execute immediate statement>.	125
19.14	<dynamic declare cursor>.	126
19.15	<allocate cursor statement>	126
19.16	<dynamic open statement>	127
19.17	<dynamic fetch statement>	128

19.19	<dynamic close statement>	129
19.20	<dynamic delete statement: positioned>	129
19.21	<dynamic update statement: positioned>	129
19.22	<preparable dynamic delete statement: positioned>	130
19.23	<preparable dynamic update statement: positioned>	130
22	Diagnostics management.....	130
22.1	<get diagnostics statement>.....	130
23	Status codes.....	131
23.1	SQLSTATE.....	131
24	Conformance.....	131
24.3	Implied feature relationships of SQL/Foundation	131
Annex A (informative)	SQL Conformance Summary	132
Annex B (informative)	Implementation-defined elements	133
Annex C (informative)	Implementation-dependent elements	134
Annex E (informative)	Incompatibilities with ISO/IEC 9075-2:1999	135
Annex F (informative)	SQL feature taxonomy	135

Tables

Table	Page
32	SQLSTATE class and subclass values. 131
34	Implied feature relationships of SQL/Foundation. 131
34	Implied feature relationships of SQL/Foundation. 132
4	Feature taxonomy for optional features. 135

Information technology — Database languages — SQL —

Part 2:

Foundation (SQL/Foundation)

TECHNICAL CORRIGENDUM 2

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 functional dependencies in <joined table> were omitted.*

Insert the following after the 5th paragraph:

If $A \rightarrow B$ 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 $CA \rightarrow CB$ 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 $A \rightarrow B$ 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 $CA \rightarrow CB$ 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 ...”).

4.27.5 Result sets returned by SQL-invoked procedures

1. *Rationale: New subclause for result sets returned by SQL-invoked procedures.*

Insert the following new subclause:

4.27.5 Result sets returned by SQL-invoked procedures

NOTE 48.1 — Subclause 3.1.6, “Definitions provided in Part 2”, gives definitions of the terms result set, result set sequence, returned result set, and with-return cursor, used in this Subclause.

An invocation of an SQL-invoked procedure *SIP1* might bring into existence a result set sequence *RSS*. *RSS* consists of the result sets of with-return cursors opened by *SIP1* and remaining open when *SIP1* terminates, placed in the order in which those result sets are created during the execution of *SIP1*.

NOTE 48.2 — If the same cursor is opened more than once during the execution of *SIP1*, then it is the last opening that is considered to create the result set, even if the result set in question is identical to that created by some earlier opening.

RSS is available, by executing a particular form of <allocate cursor statement>, only to the invoker *INV* of *SIP1*. If *SIP1* is invoked by an SQL-invoked procedure *SIP2*, then *INV* is *SIP2*. If *SIP1* is invoked by an externally-invoked procedure *EIP*, then *INV* is the SQL-client module containing *EIP*. Otherwise, *RSS* is not available.

NOTE 48.3 — Only the immediate invoker is considered. For example, if an externally-invoked procedure *EIP* executes a <call statement> invoking an SQL-invoked procedure *SIP3* that invokes *SIP1*, then the result set sequence returned by *SIP1* is available only to *SIP3*, until either *SIP3* returns control to *EIP* or another invocation of *SIP1* by *SIP3* is given before *SIP3* returns. There is no mechanism whereby *SIP3* can return *SIP1*'s result set sequence to the invoker of *SIP3*, even if *SIP3* is defined to be able to return a result set sequence.

The invocation of *SIP1* by *INV* destroys any existing result set sequence that might have arisen from some previous invocation of *SIP1* by *INV*. All result set sequences available to *INV* are destroyed when *INV* terminates.

A returned result set consists of a sequence of rows called the *constituent rows* and an *initial cursor* position.

The constituent rows and initial cursor position of each returned result set *RS* in *RSS* are determined when *SIP1* returns to *INV*. If the with-return cursor *C* for *RS* is scrollable, then the constituent rows of *RS* are those of the result set of *C* as it exists when *SIP1* terminates; otherwise, the constituent rows are as for scrollable cursors, except that rows preceding the current cursor position of *C* are excluded. If *C* is scrollable, then the initial cursor position of *RS* is the position of *C* when *SIP1* terminates; otherwise, the initial cursor position of *RS* is before the first row.

NOTE 48.4 — The result set of *C* as it exists when *SIP1* terminates might differ from that generated when *C* was opened, if, for example, any <delete statement: positioned>s or <update statement: positioned>s are executed by *SIP1* before it terminates.

The maximum number of returned result sets that may form a result set sequence is specified by the <dynamic result sets characteristic> contained in the <SQL-invoked routine> defining *SIP1*. If the actual number of with-return cursors that remain open when *SIP1* returns is greater than the maximum number of returned result sets specified in the <dynamic result sets characteristic> clause, then a warning condition is raised. It is implementation-dependent whether or not result sets whose positions are greater than that maximum number are returned.

4.32 Cursors

4.32.1 General description of cursors

1. *Rationale: Clarify definition of result sets.*

Replace the 7th paragraph with:

A cursor in the open state identifies a result set and a position relative to the ordering of that result set. If the <declare cursor> does not contain an <order by clause>, or contains an <order by clause> that does not specify the order of the rows completely, then the rows of the result set have an order that is defined only to the extent that the <order by clause> specifies an order and is otherwise implementation-dependent.

NOTE 50.1 — A definition of “result set” is given in [Subclause 3.1.6](#), “Definitions provided in Part 2”.

2. *Rationale: Correct the definition of updatable cursor.*

Replace the 10th paragraph with:

A cursor is either *updatable* or *not updatable*. If FOR UPDATE OF is specified for the cursor, or if the table identified by the cursor is simply updatable and FOR READ ONLY, SCROLL, and ORDER BY are not specified for the cursor, then the cursor is updatable; otherwise, the cursor is not updatable. The operations of update and delete are permitted for updatable cursors, subject to constraining Access Rules and Conformance Rules.

4.32.2 Operations on and using cursors

1. *Rationale: Clarify definition of returned result sets.*

Replace the 11th paragraph with:

A <declare cursor> *DC* that specifies WITH RETURN defines a with-return cursor. The <cursor specification> *CR* contained in *DC* defines a returned result set. A with-return cursor, if declared in an SQL-invoked procedure and in the open state when the procedure returns to its invoker, yields a returned result set that can be accessed by the invoker of the procedure that generates it.

NOTE 50.2 — Definitions of “returned result set” and “with-return cursor” are given in [Subclause 3.1.6](#), “Definitions provided in Part 2”.

4.33 SQL-statements

4.33.3 SQL-statements and SQL-data access indication

1. *Rationale: All queries read SQL data, not just <subquery>s.*

Replace the 2nd paragraph with:

The following SQL-statements possibly read SQL-data:

- SQL-data statements other than SQL-data change statements, <free locator statement>, and <hold locator statement>.
- SQL-statements that contain a <query expression> and are not SQL-statements that possibly modify SQL-data.

4.33.4 SQL-statements and transaction states

1. *Rationale: SQL-transaction statements can initiate SQL-transactions.*

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

The following SQL-statements are transaction-initiating SQL-statements, Thus, if there is no current SQL-transaction, and a statement of this class is executed, then an SQL-transaction is initiated:

- All SQL-schema statements
- The following SQL-transaction statements:
 - <start transaction statement>.
 - <savepoint statement>.
 - <commit statement>.
 - <rollback statement>.
- The following SQL-data statements:
 - <open statement>.
 - <close statement>.
 - <fetch statement>.
 - <select statement: single row>.
 - <insert statement>.
 - <delete statement: searched>.

- <delete statement: positioned>.
- <update statement: searched>.
- <update statement: positioned>.
- <merge statement>.
- <allocate cursor statement>.
- <dynamic open statement>.
- <dynamic close statement>.
- <dynamic fetch statement>.
- <direct select statement: multiple rows>.
- <dynamic single row select statement>.
- <dynamic delete statement: positioned>.
- <preparable dynamic delete statement: positioned>.
- <dynamic update statement: positioned>.
- <preparable dynamic update statement: positioned>.
- <free locator statement>.
- <hold locator statement>.

— The following SQL-dynamic statements

- <describe input statement>.
- <describe output statement>.
- <allocate descriptor statement>.
- <deallocate descriptor statement>.
- <get descriptor statement>.
- <set descriptor statement>.
- <prepare statement>.
- <deallocate prepared statement>.

With the exception of <start transaction statement>, every transaction-initiating SQL-statement *implicitly initiates a transaction* if there is no current SQL-transaction when it is executed, in which case the execution of the initiating statement is included in the initiated SQL-transaction. In the case of <start transaction statement>, transaction initiation is the primary effect of executing the statement itself.

Whether or not an <execute immediate statement> starts a transaction depends on the content of the <SQL statement variable> referenced by the <execute immediate statement> at the time it is executed. Whether or not an <execute statement> starts a transaction depends on the content of the <SQL statement variable> referenced by the <prepare statement> at the time the prepared statement referenced by the <execute statement> was prepared. In both cases, if the content of the <SQL statement variable> was a transaction-

initiating SQL-statement, then the <execute immediate statement> or <execute statement> is treated as a transaction-initiating statement; otherwise, it is not treated as a transaction-initiating statement.

The following SQL-statements are not transaction-initiating SQL-statements, Thus, if there is no current SQL-transaction, then executing a statement of this class does not change that state of affairs.

- All SQL-transaction statements except <start transaction statement>s, <savepoint statement>s, <commit statement>s, and <rollback statement>s.
- All SQL-connection statements.
- All SQL-session statements.
- All SQL-diagnostics statements.
- SQL embedded exception declarations.
- The following SQL-data statements:
 - <temporary table declaration>.
 - <declare cursor>.
 - <dynamic declare cursor>.
 - <dynamic select statement>.

2. *Rationale: All queries need a transaction to have been started, not just <subquery>s.*

Replace the final (6th) paragraph with:

If an <SQL control statement> causes the evaluation of a <query expression> and there is no current SQL-transaction, then an SQL-transaction is initiated before evaluation of the <query expression>.

4.33.5 SQL-statement atomicity and statement execution contexts

1. *Rationale: The evaluation of a <subquery> does not require an atomic execution context*

Replace the 6th paragraph with:

The statement execution context brought into existence by the execution of an atomic SQL-statement is an *atomic execution context*.

4.34 Basic security model

4.34.1 Authorization identifiers

1. *Rationale: Inappropriate angle-brackets, implementation-dependent mappings of user identifiers are of no possible use, so they must be implementation-defined, and use of database system instead of SQL-implementation.*

Replace the 1st paragraph with:

An authorization identifier identifies a set of privileges. An authorization identifier is either a user identifier or a role name. A user identifier represents a user of the SQL-implementation. Any mapping of user identifiers to operating system users is implementation-defined. A role name represents a role.

4.34.1.1 SQL-session authorization identifiers

1. *Rationale: Inappropriate angle-brackets.*

Replace the 1st paragraph with:

An SQL-session has a user identifier called the *SQL-session user identifier*. When an SQL-session is initiated, the SQL-session user identifier is determined in an implementation-defined manner, unless the session is initiated using a <connect statement>. The value of the SQL-session user identifier can never be the null value. The SQL-session user identifier can be determined by using SESSION_USER.

4.34.2 Privileges

1. *Rationale: Define view component privilege descriptors. Only <grant privilege statement> uses GRANT OPTION.*

Replace the 1st paragraph with:

A privilege authorizes a given category of <action> to be performed by a specified <authorization identifier> on a specified base table, view, view component, column, domain, character set, collation, transliteration, user-defined type, table/method pair, SQL-invoked routine, or sequence generator.

2. *Rationale: Define view component privilege descriptors.*

Replace the 1st bullet item in the 2nd paragraph with:

- The identification of the base table, view, view component, column, domain, character set, collation, transliteration, user-defined type, table/method pair, SQL-invoked routine, or sequence generator that the descriptor describes.

3. *Rationale: Define view component privilege descriptors.*

Replace the 4th and 5th paragraphs with:

A privilege descriptor with an <action> of INSERT, UPDATE, DELETE, SELECT, TRIGGER, or REFERENCES is called a *table privilege descriptor* and identifies the existence of a privilege on the table or view component identified by the privilege descriptor. If a table privilege descriptor identifies a view component, the privilege descriptor is called a *view component table privilege descriptor*.

A privilege descriptor with an <action> of SELECT (<column name list>), INSERT (<column name list>), UPDATE (<column name list>), or REFERENCES (<column name list>) is called a *column privilege descriptor* and identifies the existence of a privilege on the columns in the table or view component identified by the privilege descriptor. If a column privilege descriptor identifies a view component, the privilege descriptor is called a *view component column privilege descriptor*.

4. *Rationale: Define view component privilege descriptors.*

Insert the following paragraph after the 10th paragraph:

A *view privilege dependency descriptor* is a descriptor that includes two privilege descriptors, called the *supporting privilege descriptor* and the *dependent privilege descriptor*. A view privilege dependency descriptor is a record that an INSERT, UPDATE, or DELETE privilege of a view, or a column of a view, is directly dependent on another privilege.

5. *Rationale: Only <grant privilege statement> uses GRANT OPTION.*

Replace the 11th paragraph with:

A grantable privilege is a privilege, associated with a schema object, that may be granted by a <grant privilege statement>. The WITH GRANT OPTION clause of a <grant privilege statement> specifies whether the <authorization identifier> recipient of a privilege (acting as a grantor) may grant it to others.

4.34.3 Roles

1. *Rationale: Various editorial corrections and additions, and removal of an unnecessary definition.*

Replace the entire Subclause with:

A role, identified by a role name, is, like a user, a potential grantee and grantor of privileges and other roles. Also like a user, a role can additionally own schemas and other objects.

A role is created by executing a <role definition> and destroyed by executing a <drop role statement>.

A role is granted to one or more authorization identifiers by executing a <grant role statement>, thus conferring on the grantees all the privileges of that role. The granting of a role to an authorization identifier *A* is called a *role authorization* (for *A*).

The privileges of a role with role name *R* are the union of the privileges whose grantee is *R* and the sets of privileges for the role names defined by the role authorizations for *R*. Cycles of role authorizations are prohibited.

The WITH ADMIN OPTION clause of the <grant role statement> for role *R* specifies that each grantee may grant *R* to others, revoke *R* from others, and destroy *R*.

Each role authorization is described by a *role authorization descriptor*. A role authorization descriptor includes:

- The role name of the role.
- The authorization identifier of the grantor.
- The authorization identifier of the grantee.
- An indication of whether or not the role authorization is grantable.

4.34.4 Security model definitions

1. *Rationale: Insertion of one missing definition and simplification of others.*

Replace the entire subclause with:

A role *R* is *applicable for* an authorization identifier *A* if there exists a role authorization descriptor whose role name is *R* and whose grantee is PUBLIC, or *A*, or an applicable role for *A*.

A privilege *P* is *applicable for* an authorization identifier *A* if its grantee is PUBLIC, or *A*, or an applicable role for *A*.

NOTE 52.1 — *applicable for* is a persistent relationship between persistent objects. Thus, it in no way depends on any SQL-session.

An authorization identifier is *enabled* if it is the current user identifier, the current role name, or a role name that is applicable for the current role name.

A privilege *P* is *current* if *P* is applicable for an enabled authorization identifier.

NOTE 52.2 — *enabled* and *current* apply to (transient) elements of the current SQL-session context.

4.37 SQL-sessions

4.37.3 SQL-session properties

1. *Rationale: Clarify how enduring transaction characteristics are altered.*

Replace the 10th and 11th paragraphs with:

An SQL-session has the following enduring characteristics:

— *enduring transaction characteristics*

Each of the enduring characteristics can be altered at any time in an SQL-session by executing an appropriate <set session characteristics statement>.

2. *Rationale: Avoid misleading references to “current transaction”.*

In the 12th paragraph, replace the 7th dashed item with:

— The current access mode.

3. *Rationale: Avoid misleading references to “current transaction”.*

In the 12th paragraph, replace the 9th dashed item with:

— The current isolation level.

4. *Rationale: Add result set sequences to SQL-session context.*

In the 12th paragraph, insert the following after the 24th bullet:

— Each currently available result set sequence *RSS*, along with the specific name of an SQL-invoked procedure *SIP* and the name of the invoker of *SIP* for the invocation causing *RSS* to be brought into existence.

NOTE 54.1 — Result set sequences are defined in Subclause 4.27.5, “Result sets returned by SQL-invoked procedures”.

4.38 Triggers

4.38.2 Trigger execution

1. *Rationale: Remove incorrect information about trigger execution contexts.*

Delete the 2nd paragraph.

2. *Rationale: Correct the information about where transitions are specified.*

Replace the 17th paragraph with:

Let $PSCN$ be the number of elements in PSC . A state change $SC_{i,j}$, 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 that SSC_i does not already contain a state change corresponding to $SC_{i,j}$. Transitions are added to $SC_{i,j}$ as specified by the General Rules of Subclause 14.16, “Effect of deleting rows from base tables”, Subclause 14.19, “Effect of inserting tables into base tables”, and Subclause 14.22, “Effect of replacing rows in base tables”.

5 Lexical elements

5.1 <SQL terminal character>

1. *Rationale: Correct character being defined.*

Replace the production for <right bracket> with:

```
<right bracket> ::=  
    ]
```

5.2 <token> and <separator>

1. *Rationale: Explicitly make lower case “u” permissable.*

Insert the following Syntax Rule:

- 14.1) In a <Unicode delimited identifier>, the introductory 'U' may be represented either in upper case (as 'U') or in lower case (as 'u').

2. *Rationale: Editorial.*

Replace Syntax Rules 15) and 16) with:

- 15) <Unicode escape character> shall be a single character from the source language character set other than a <hexit>, <plus sign>, <quote>, <double quote>, or <white space>.
- 16) If the source language character set contains <reverse solidus>, then let *DEC* be <reverse solidus>; otherwise, let *DEC* be an implementation-defined character from the source language character set that is not a <hexit>, <plus sign>, <quote>, <double quote>, or <white space>.

5.3 <literal>

1. *Rationale: Explicitly make lower case “u” permissable.*

Insert the following Syntax Rule:

- 2.1) In a <Unicode character string literal>, the introductory 'U' may be represented either in upper case (as 'U') or in lower case (as 'u').

2. *Rationale: Explicitly make lower case “x” permissable.*

Insert the following Syntax Rule:

- 3.1) In a <binary string literal>, the introductory 'X' may be represented either in upper case (as 'X') or in lower case (as 'x').

3. *Rationale: Explicitly make lower case “n” permissable.*

Insert the following Syntax Rule:

- 5.1) In a <national character string literal>, the introductory 'N' may be represented either in upper case (as 'N') or in lower case (as 'n').

4. *Rationale: Explicitly make lower case “e” permissable.*

Insert the following Syntax Rule:

- 19.1) In an <approximate numeric literal>, the exponent indicator 'E' may be represented either in upper case (as 'E') or in lower case (as 'e').

5.4 Names and identifiers

1. *Rationale: Repair an omission and remove unnecessary definition of <local qualified name>.*

Replace Syntax Rule 7) with:

- 7) Let *CN* be a <cursor name>. *CN* shall be contained in an <SQL-client module definition> whose <module contents> contain a <declare cursor> or <dynamic declare cursor> whose <cursor name> is *CN*.

2. *Rationale: Correct misspelling.*

Replace Syntax Rule 9) with:

- 9) Two <user-defined type name>s are equivalent if and only if they have equivalent <qualified identifier>s and equivalent <schema name>s, regardless of whether the <schema name>s are implicit or explicit.

3. *Rationale: Determine the implicit schema name of a constraint name correctly.*

Replace Syntax Rule 13) with:

- 13) If a <schema qualified name> *SQLN* other than a <transcoding name> does not contain a <schema name>, then

Case:

- a) If any of the following is true:
 - i) *SQLN* is immediately contained in a <collation name> that is not immediately contained in a <collation definition> or in a <drop collation statement>.
 - ii) *SQLN* is immediately contained in a <transliteration name> that is not immediately contained in a <transliteration definition> or in a <drop transliteration statement>. then <schema name> INFORMATION_SCHEMA is implicit.
- b) If *SQLN* is immediately contained in a <constraint name> that is contained in a <table definition> or an <alter table statement>, then the explicit or implicit <schema name> of the <table name> of the table identified by the <table definition> or <alter table statement> is implicit.
- c) If *SQLN* is immediately contained in a <constraint name> that is contained in a <domain definition> or an <alter domain statement>, then the explicit or implicit <schema name> of the <domain name> of the domain identified by the <domain definition> or <alter domain statement> is implicit.

- d) Otherwise,

Case:

- i) If *SQLN* is contained, without an intervening <schema definition>, in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default <unqualified schema name> for the SQL-session is implicit.

- ii) If *SQL* is contained in a <schema definition>, then the <schema name> that is specified or implicit in the <schema definition> is implicit.
- iii) Otherwise, the <schema name> that is specified or implicit for the <SQL-client module definition> is implicit.

4. *Rationale: Prohibit the use of LOCAL in places where it makes no sense.*

Replace Syntax Rule 28) with:

- 28) In a <descriptor name>, <extended statement name>, or <extended cursor name>, if a <scope option> is not specified, then a <scope option> of LOCAL is implicit. If a <scope option> is contained in an <SQL-schema statement> then it shall not contain LOCAL.

5. *Rationale: Clarification of the method of identification of dynamic objects.*

Replace General Rule 31) with:

- 31) If a prepared statement *PSX* is created in SQL-session *SS* by executing a <prepare statement> *PS1* that contains an <extended statement name> *ESN1* whose value at the time of execution is *V*, then, for as long as it exists, *PSX* can be identified by an <extended statement name> *ESN2* in an <SQL procedure statement> *PS2* executed in *SS* if the value of *ESN2* at the time of execution is *V* and the <scope option> of *ESN2* is the same as the <scope option> of *ESN1*. If the <scope option> of *ESN1* is LOCAL, then *ESN2* identifies *PSX* only if *PS2* is contained in the same <SQL-client module definition> as *PS1*.

6. *Rationale: Clarification of the method of identification of dynamic objects.*

Replace General Rule 34) with:

- 34) If a cursor *CSR* is created in SQL-session *SS* by executing an <allocate cursor statement> *ACS* that contains an <extended statement name> *ESN1* whose value at the time of execution is *V*, then, for as long as it exists, *CSR* can be identified by an <extended statement name> *ESN2* in an <SQL procedure statement> *PS2* executed in *SS* if the value of *ESN2* at the time of execution is *V* and the <scope option> of *ESN2* is the same as the <scope option> of *ESN1*. If the <scope option> of *ESN1* is LOCAL, then *ESN2* identifies *CSR* only if *PS2* is contained in the same <SQL-client module definition> as *ACS*.

7. *Rationale: Clarification of the method of identification of dynamic objects.*

Replace General Rule 35) with:

- 35) If an SQL descriptor area *SDA* is created in SQL-session *SS* by executing an <allocate descriptor statement> *ADS* that contains an <extended statement name> *ESN1* whose value at the time of execution is *V*, then, for as long as it exists, *SDA* can be identified by an <extended statement name> *ESN2* in an <SQL procedure statement> *PS2* executed in *SS* if the value of *ESN2* at the time of execution is *V* and the <scope option> of *ESN2* is the same as the <scope option> of *ESN1*. If the <scope option> of *ESN1* is LOCAL, then *ESN2* identifies *SDA* only if *PS2* is contained in the same <SQL-client module definition> as *ADS*.

6 Scalar expressions

6.1 <data type>

1. *Rationale: <char length units> is not appropriate to a <binary large object string type>.*

Insert the following Syntax Rule:

- 5.1) A <binary large object string type> shall not contain a <char length units>.

2. *Rationale: Correct name of the type.*

Replace Syntax Rule 6) with:

- 6) If <char length units> is specified, then the character repertoire of the explicit or implicit character set of the character string type shall be UCS.

3. *Rationale: <char length units> is not appropriate to a <binary large object string type>.*

Replace Syntax Rule 7) with:

- 7) If <data type> contains a <character string type> or <character large object type> and does not contain a <char length units>, then CHARACTERS is implicit.

6.4 <value specification> and <target specification>

1. *Rationale: Omission of <target array reference>.*

Replace Syntax Rule 7) with:

- 7) A <target specification>, <target array reference>, or <simple target specification> that is a <column reference> shall be a new transition variable column reference.

2. *Rationale: Inappropriate use of the definite article, omission of array element, and ambiguity.*

Replace General Rule 3) with:

- 3) A <target specification> specifies a target that is a host parameter, an output SQL parameter, a column of a new transition variable, an element of a target whose declared type is an array type, a parameter used in a dynamically prepared statement, or a host variable, according to whether the <target specification> is a <host parameter specification>, an <SQL parameter reference>, a <column reference>, a <target array element specification>, a <dynamic parameter specification>, or an <embedded variable specification>, respectively.

3. *Rationale: Omission of embedded variable, and presence of non-normative text.*

Replace General Rule 13) with:

- 13) A <simple target specification> specifies a target that is a host parameter, an output SQL parameter, a column of a new transition variable, or a host variable, according to whether the <simple target specification> is a <host parameter specification>, an <SQL parameter reference>, a <column reference>, or an <embedded variable name>, respectively.

NOTE 91.1 — A <simple target specification> can never be assigned the null value.

4. *Rationale: The word “specific” is not necessary.*

Replace Conformance Rule 5) with:

- 5) Without Feature F611, “Indicator data types”, in conforming SQL language, the declared types of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

6.6 <identifier chain>

1. *Rationale: SQL-92 did not permit common column names to be qualified with a range variable.*

Replace Syntax Rules 8) b) iv) to vi) inclusive with the following:

- 8) ...
 - b) ...
 - iv) If $N = 2$ and PIC_1 is equivalent to an exposed <correlation name> that is in scope, then let EN be the exposed <correlation name> that is equivalent to PIC_1 and has innermost scope. For every column C in the table associated with EN whose <column name> is equivalent to I_2 and is not a common column name, PIC_2 is a candidate basis of IC , the scope of PIC_2 is the scope of EN , and the referent of PIC_2 is C .
 - v) If $N > 2$ and PIC_1 is equivalent to an exposed <correlation name> that is in scope, then let EN be the exposed <correlation name> that is equivalent to PIC_1 and has innermost scope. For every refinable column C in the table associated with EN whose <column name> is equivalent to I_2 and is not a common column name, PIC_2 is a candidate basis of IC , the scope of PIC_2 is the scope of EN , and the referent of PIC_2 is C .
 - vi) If $N = 2, 3$ or 4 , and if PIC_{N-1} is equivalent to an exposed <table or query name> that is in scope, then let EN be the exposed <table or query name> that is equivalent to PIC_{N-1} and has the innermost scope. For every column C in the table associated with EN whose <column name> is equivalent to I_N and is not a common column name, PIC_N is a candidate basis of IC , the scope of PIC_N is the scope of EN , and the referent of PIC_N is C .

6.7 <column reference>

1. *Rationale: Remove misleading redundant text and reference syntax element.*

Replace Syntax Rule 6) with:

- 6) A <column reference> contained in a <query specification> is a *queried column reference*.

2. *Rationale: A <query primary> cannot be a <joined table>.*

Replace Syntax Rule 7) a) with:

- 7) ...
 - a) The *qualifying query* of QCR is the <query specification> that simply contains the <from clause> that simply contains the <table reference> that defines the qualifying table of QCR .

3. *Rationale: A <query primary> cannot be a <joined table> and <column reference>s in a JOIN ON <search condition> must be “ordinary”.*

Replace Syntax Rule 7) b) i) with:

- 7) ...
 - b) ...
 - i) If QQ is not grouped, or if QCR is contained in the <where clause> simply contained in QQ, then QCR is an *ordinary column reference*.

6.9 <set function specification>

1. *Rationale: “<subquery>” should be “<query expression>”.*

Replace Syntax Rule 1) with:

- 1) If <aggregate function> specifies a <general set function>, then the <value expression> simply contained in the <general set function> shall not contain a <set function specification> or a <query expression>.
2. *Rationale: “<subquery>” should be “<query expression>”.*

Replace Syntax Rule 2) with:

- 2) If <aggregate function> specifies <binary set function>, then neither the <dependent variable expression> nor the <independent variable expression> simply contained in the <binary set function> shall contain a <set function specification> or a <query expression>.

6.10 <window function>

1. *Rationale: “<subquery>” should be “<query expression>”.*

Replace Syntax Rule 4) with:

- 4) OF shall not contain an outer reference or a <query expression>.

6.12 <cast specification>

1. *Rationale: Need to specify the returned value.*

Replace General Rules 1) and 2) with:

- 1) Let *CS* be the <cast specification>. If the <cast operand> is a <value expression> *VE*, then let *SV* be the value of *VE*.
- 2) Case:
 - a) If the <cast operand> specifies NULL, then the result of *CS* is the null value and no further General Rules of this Subclause are applied.
 - b) If the <cast operand> specifies an <empty specification>, then the result of *CS* is an empty collection of declared type *TD* and no further General Rules of this Subclause are applied.
 - c) If *SV* is the null value, then the result of *CS* is the null value and no further General Rules of this Subclause are applied.

2. *Rationale: Use the correct symbol.*

Replace General Rule 3) a) with:

- 3) ...
 - a) If *TD* is a supertype of *SD*, then *TV* is *SV*.

3. *Rationale: Correct use of values in syntactic substitutions.*

Delete General Rule 5) b).

4. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 5) c) with:

- 5) ...
 - c) For *i* varying from 1 (one) to *SC*, the following <cast specification> is applied:

CAST (*VE*[*i*] AS *ETD*)

yielding value *TVE_i*.

5. *Rationale: Correct use of values in syntactic substitutions.*

Delete General Rules 6) b) and 6) c).

6. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 6) d) and 6) e) with:

- 6) ...
 - d) If *TD* is an array type, then let *TC* be the maximum cardinality of *TD*.

Case:

i) If *SC* is greater than *TC*, then an exception condition is raised: *data exception — array data, right truncation*.

ii) Otherwise, *TV* is the array resulting from the execution of:

```
ARRAY (
  ( SELECT CAST ( M.E AS ETD ) FROM UNNEST ( VE ) AS M(E) )
)
```

e) If *TD* is a multiset type, then *TV* is the multiset resulting from the execution of:

```
MULTISET ( ( SELECT CAST ( M.E AS ETD ) FROM UNNEST ( VE ) AS M(E) ) )
```

7. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 7) with:

7) If *SD* is a <row type>, then *TV* is the row resulting from the execution of:

```
ROW ( CAST ( VE.FSD1 AS TFTD1 ), CAST ( VE.FSD2 AS TFTD2 ), ...
      CAST ( VE.FSDDSD AS TFTDDSD ) )
```

8. *Rationale: Delete ill-formed rule and make all invalid character strings for CAST to datetime and interval types result in the same exception.*

Delete General Rule 13) a) ii).

9. *Rationale: Delete ill-formed rule and make all invalid character strings for CAST to datetime and interval types result in the same exception.*

Replace General Rule 13) b) with:

13) ...

b) If *SD* is the datetime data type DATE, then *TV* is SV.

10. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 13) d) with:

13) ...

d) If *SD* is the datetime data type TIMESTAMP WITH TIME ZONE, then *TV* is computed by:

```
CAST ( CAST ( VE AS TIMESTAMP WITHOUT TIME ZONE ) AS DATE )
```

11. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 15) a) ii) with:

15) ...

- a) ...
- ii) If the rules for <literal> or for <unquoted time string> in Subclause 5.3, “<literal>”, can be applied to SV to determine a valid value of the data type TIME(TSP) WITH TIME ZONE, then TV is:

```
CAST ( CAST ( VE AS TIME(TSP) WITH TIME ZONE )
        AS TIME(TSP) WITHOUT TIME ZONE )
```

12. *Rationale: Delete ill-formed rule and make all invalid character strings for CAST to datetime and interval types result in the same exception.*

Replace General Rule 15) a) iii) and General Rule 15) a) iv) with:

- 15) ...
- a) ...
- iii) Otherwise, an exception condition is raised: *data exception — invalid datetime format.*

13. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 15) e) with:

- 15) ...
- e) If SD is TIMESTAMP WITH TIME ZONE, then TV is:

```
CAST ( CAST ( VE AS TIMESTAMP(TSP) WITHOUT TIME ZONE )
        AS TIME(TSP) WITHOUT TIME ZONE )
```

14. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 16) a) ii) with:

- 16) ...
- a) ...
- ii) If the rules for <literal> or for <unquoted time string> in Subclause 5.3, “<literal>”, can be applied to SV to determine a valid value of the data type TIME(TSP) WITHOUT TIME ZONE, then TV is:

```
CAST ( CAST ( VE AS TIME(TSP) WITHOUT TIME ZONE )
        AS TIME(TSP) WITH TIME ZONE )
```

15. *Rationale: Delete ill-formed rule and make all invalid character strings for CAST to datetime and interval types result in the same exception.*

Replace General Rule 16) a) iii) and General Rule 16) a) iv) with:

- 16) ...

- a) ...
- iii) Otherwise, an exception condition is raised: *data exception — invalid datetime format*.

16. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 16) e) with:

- 16) ...
- e) If *SD* is `TIMESTAMP WITHOUT TIME ZONE`, then *TV* is:

`CAST (CAST (VE AS TIMESTAMP(TSP) WITH TIME ZONE)
AS TIME(TSP) WITH TIME ZONE)`

17. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 17) a) ii) with:

- 17) ...
- a) ...
- ii) If the rules for `<literal>` or for `<unquoted time string>` in [Subclause 5.3](#), “`<literal>`”, can be applied to *SV* to determine a valid value of the data type `TIMESTAMP(TSP) WITH TIME ZONE`, then *TV* is:

`CAST (CAST (VE AS TIMESTAMP(TSP) WITH TIME ZONE)
AS TIMESTAMP(TSP) WITHOUT TIME ZONE)`

18. *Rationale: Delete ill-formed rule and make all invalid character strings for `CAST` to datetime and interval types result in the same exception.*

Replace General Rule 17) a) iii) and General Rule 17) a) iv) with:

- 17) ...
- a) ...
- iii) Otherwise, an exception condition is raised: *data exception — invalid datetime format*.

19. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 17) d) with:

- 17) ...
- d) If *SD* is `TIME WITH TIME ZONE`, then *TV* is:

`CAST (CAST (VE AS TIMESTAMP(TSP) WITH TIME ZONE)
AS TIMESTAMP(TSP) WITHOUT TIME ZONE)`

20. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 18) a) ii) with:

18) ...

a) ...

- ii) If the rules for <literal> or for <unquoted time string> in [Subclause 5.3](#), “<literal>”, can be applied to *SV* to determine a valid value of the data type `TIMESTAMP(TSP) WITHOUT TIME ZONE`, then *TV* is:

```
CAST ( CAST ( VE AS TIMESTAMP(TSP) WITHOUT TIME ZONE )
      AS TIMESTAMP(TSP) WITH TIME ZONE )
```

21. *Rationale: Delete ill-formed rule and make all invalid character strings for CAST to datetime and interval types result in the same exception.*

Replace General Rule 18) a) iii) and General Rule 18) a) iv) with:

18) ...

a) ...

- iii) Otherwise, an exception condition is raised: *data exception — invalid datetime format*.

22. *Rationale: Correct use of values in syntactic substitutions.*

Replace General Rule 18) b) and 18) c) with:

18) ...

- b) If *SD* is a date, then *TV* is computed by:

```
CAST ( CAST ( VE AS TIMESTAMP WITHOUT TIME ZONE )
      AS TIMESTAMP WITH TIME ZONE )
```

- c) If *SD* is `TIME WITHOUT TIME ZONE`, then *TV* is:

```
CAST ( CAST ( VE AS TIMESTAMP WITHOUT TIME ZONE )
      AS TIMESTAMP WITH TIME ZONE )
```

23. *Rationale: Delete ill-formed rule and make all invalid character strings for CAST to datetime and interval types result in the same exception.*

Replace General Rule 19) b) ii) with:

19) ...

b) ...

- ii) Otherwise, an exception condition is raised: *data exception — invalid interval format*.

24. *Rationale: Need to specify the returned value.*

Insert the following General Rule:

21.1) The result of CS is TV.

6.13 <next value expression>

1. *Rationale: Define the data type of the <next value expression>.*

Insert the following Syntax Rule:

- 3) The declared type of <next value expression> is the data type described by the data type descriptor included in the sequence generator descriptor identified by <sequence generator name>.

6.27 <numeric value function>

1. *Rationale: CHARACTER_LENGTH cannot support binary string arguments.*

In the Format, replace the production for <char length expression> with:

<char length expression> ::=

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

2. *Rationale: General Rule 1) establishes the result of a <numeric value function> to be the null value, if any of the input values are the null value. However, subsequent General Rules will overwrite this result.*

Replace General Rule 1) with:

- 1) If the value of one or more <string value expression>s, <datetime value expression>s, <interval value expression>s, and <collection value expression>s that are simply contained in a <numeric value function> is the null value, then the result of the <numeric value function> is the null value and no further General Rules of this Subclause are applied.
3. *Rationale: Correct the definition of MOD function. The functionality was intended to be aligned with that of the FORTRAN MOD function and the C fmod function and % operator.*

Replace the lead text of General Rule 9) c) with:

9) ...

- c) Otherwise, the result is the unique exact numeric value *R* with scale 0 (zero) such that all of the following are true:

6.29 <string value function>

1. *Rationale: Remove test on value from Syntax Rules.*

Delete Syntax Rule 7) d).

7 Query expressions

7.1 <row value constructor>

1. *Rationale: Editorial. Correct reference to BNF non-terminal*

Replace the production for <contextually typed row value constructor> with:

```
<contextually typed row value constructor> ::=
    <common value expression>
  | <boolean value expression>
  | <contextually typed value specification>
  | <left paren> <contextually typed value specification> <right paren>
  | <left paren> <contextually typed row value constructor element> <comma>
    <contextually typed row value constructor element list> <right paren>
  | ROW <left paren> <contextually typed row value constructor element list> <right paren>
```

7.6 <table reference>

1. *Rationale: Allow nested parentheses around <joined table>.*

In the Format, replace the production for <table primary> with:

```
<table primary> ::=
    <table or query name> [ [ AS ] <correlation name>
      [ <left paren> <derived column list> <right paren> ] ]
  | <derived table> [ AS ] <correlation name>
      [ <left paren> <derived column list> <right paren> ]
  | <lateral derived table> [ AS ] <correlation name>
      [ <left paren> <derived column list> <right paren> ]
  | <collection derived table> [ AS ] <correlation name>
      [ <left paren> <derived column list> <right paren> ]
  | <table function derived table> [ AS ] <correlation name>
      [ <left paren> <derived column list> <right paren> ]
  | <only spec> [ [ AS ] <correlation name>
      [ <left paren> <derived column list> <right paren> ] ]
  | <parenthesized joined table>
```

```
<parenthesized joined table> ::=
```

<left paren> <parenthesized joined table> <right paren>
 | <left paren> <joined table> <right paren>

2. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Replace Syntax Rule 4), 5), and 6) with:

4) Let <table reference> be *TR*.

Case:

- a) If *TR* simply contains a <joined table>, let *JT* be that <joined table>.
- b) Otherwise, let *TF* be the <table factor> that is immediately contained in *TR*, and *TP* be the <table primary> that is immediately contained in *TP*.

5) If *TF* simply contains a <correlation name>, then let *RV* be that <correlation name>; otherwise, let *RV* be the <table or query name> simply contained in *TF*. *RV* is a range variable. *RV* is exposed by *TF* and *TR*.

NOTE 124 — “range variable” is defined in Subclause 4.14.6, “Operations involving tables”.

6) Case:

- a) If *TF* is contained in a <from clause> *FC* with no intervening <query expression>, then the *scope clause SC* of *TR* is the <select statement: single row> or innermost <query specification> that contains *FC*. The scope of a range variable of *TR* is the <select list>, <where clause>, <group by clause>, <having clause>, and <window clause> of *SC*, together with every <lateral derived table> that is simply contained in *FC* and is preceded by *TR*, and every <collection derived table> that is simply contained in *FC* and is preceded by *TR*, and the <join condition> of all <joined table>s contained in *SC* that contain *TR*. If *SC* is the <query specification> that is the <query expression body> of a simple table query *STQ*, then the scope of a range variable of *TR* also includes the <order by clause> of *STQ*.

NOTE 125 — “simple table query” is defined in Subclause 14.1, “<declare cursor>”.

- b) If *TF* is simply contained in a <merge statement> *MS*, then the *scope clause SC* of *TR* is *MS*. The scope of the range variable of *TF* and *TR* is the <search condition>, <set clause list>, and <merge insert value list> of *SC*.

NOTE 125.1 — Subclause 14.9, “<merge statement>”, does not allow *TR* to directly contain a <joined table>.

- c) Otherwise, the *scope clause SC* of *TR* is the outermost <joined table> that contains *TR* with no intervening <query expression>. The scope of any range variable of *TR* is the <join condition> of *SC* and of all <joined table>s contained in *SC* that contain *TR*.

3. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Replace the lead text of Syntax Rule 7) with:

- 7) Let *RV* be a range variable that is exposed by *TR*. Let *RV1* be a range variable that is exposed by a <table reference> *TR1* that has the same *scope clause* as *TR*.

4. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Replace Syntax Rule 8) with:

- 8) A <table or query name> simply contained in <table factor> *TF* has a scope clause and scope defined by *TF* if and only if the <table or query name> is exposed by *TF* and by *TR*.

5. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Replace the lead text of Syntax Rule 9) with:

- 9) If *TP* simply contains <table or query name> *TOQN*, then

6. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Replace Syntax Rule 10) with:

- 10) If *TP* simply contains <only spec> *OS* and the table identified by the <table or query name> *TN* is not a typed table, then *OS* is equivalent to *TN*.

7. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Replace Syntax Rule 12) with:

- 12) If a <derived column list> is specified in *TP*, then the number of <column name>s in the <derived column list> shall be the same as the degree of the table specified by the <derived table>, the <lateral derived table>, or the <table or query name> simply contained in *TP*, and the name of the *i*-th column of that <derived table> or <lateral derived table> or the effective name of the *i*-th column of that <table or query name> is the *i*-th <column name> in that <derived column list>.

8. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Replace Syntax Rule 14) with:

- 14) Case:

- a) If no <derived column list> is specified in *TR*, then the row type *RT* of *TR* is the row type of its simply contained <table or query name>, <derived table>, <lateral derived table>, or <joined table>.
- b) Otherwise, the row type *RT* of *TR* is described by a sequence of (<field name>, <data type>) pairs, where the <field name> in the *i*-th pair is the *i*-th <column name> in the <derived column list> and the <data type> in the *i*-th pair is the declared type of the *i*-th column of the <derived table>, <joined table>, <lateral derived table>, or of the table identified by the <table or query name> simply contained in *TR*.

9. *Rationale: Define simply updatable derived table.*

Insert the following Syntax Rule:

15.1) A <derived table> or <lateral derived table> is a *simply updatable derived table* if and only if the <query expression> simply contained in the <derived table> or <lateral derived table> is simply updatable.

10. *Rationale: Define simply updatable derived table.*

Replace Syntax Rule 17) with:

17) A <collection derived table> is not updatable and not simply updatable.

11. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Insert the following General Rule:

2.1) If *TP* immediately contains a <joined table>, then the result of *TP* is the result of that <joined table>.

12. *Rationale: Move the rule pertaining to <sample clause> from Subclause 7.15, “<subquery>” to this Subclause with appropriate rewording.*

Insert the following General Rule:

3) ...

a) ...

v) If *TF* contains outer references, then a table with identical rows is generated every time *TF* is evaluated with a given set of values for outer references.

NOTE 130.1 — “Outer reference” is defined in Subclause 6.7, “<column reference>”.

13. *Rationale: Restore support for queries of the form “SELECT . . . FROM <joined table> . . .”*

Insert the following Note after General Rule 5):

NOTE 130.1 — “table associated with RV” is defined in Subclause 4.14.6, “Operations involving tables”.

7.7 <joined table>

1. *Rationale: Corresponding join columns belong to the operands of a <joined table>, not to the result.*

Replace Syntax Rules 10), 11) and 12) with:

10) For every column *CR* of the result of the <joined table> that corresponds to a column C_1 of T_1 that is not a corresponding join column, *CR* is *possibly nullable* if any of the following conditions are true:

a) RIGHT or FULL is specified.

b) INNER, LEFT, or CROSS JOIN is specified or implicit and C_1 is possibly nullable.

- 11) For every column CR of the result of the <joined table> that corresponds to a column C_2 of T_2 that is not a corresponding join column, CR is *possibly nullable* if any of the following conditions are true:
 - a) LEFT or FULL is specified.
 - b) INNER, RIGHT, or CROSS JOIN is specified or implicit and C_2 is possibly nullable.
- 12) For every column CR of the result of the <joined table> that corresponds to a corresponding join column C_1 of T_1 and a corresponding join column C_2 of T_2 , CR is *possibly nullable* if any of the following conditions are true:
 - a) LEFT or FULL is specified and C_1 is possibly nullable.
 - b) RIGHT or FULL is specified and C_2 is possibly nullable.

2. *Rationale: Incorrect use of “subrow”, incorrect subscript, wrong side.*

Insert the following General Rule:

- 1.1) Let D_1 and D_2 be the degrees of TR_1 and TR_2 , respectively.

3. *Rationale: Incorrect use of “subrow”, incorrect subscript, wrong side.*

Replace General Rule 4) with:

- 4) Let P_1 be the collection of rows of T_1 for which there exist some row R in TR and some row R_1 in T_1 such that the values of the first D_1 fields of R are identical to the values of the corresponding fields of R_1 .

4. *Rationale: Incorrect use of “subrow”, incorrect subscript, wrong side.*

Replace General Rule 6) with:

- 6) Let X_1 be U_1 extended with D_2 columns containing the null value.

5. *Rationale: Remove incorrect use “subrow”, incorrect subscript, wrong side.*

Replace General Rule 8) b) with:

- 8) ...
 - b) Let P_2 be the collection of rows of T_2 for which there exists some row R in TR and some row R_2 in T_2 such that the values of the last D_2 fields of R are identical to the values of the corresponding fields of R_2 .

6. *Rationale: Incorrect use of “subrow”, incorrect subscript, wrong side.*

Replace General Rule 8) d) with: