X3H2-94-329
DBL:RIO-004
August, 1994

ISO

International Organization for Standardization

ANSI

American National Standards Institute

ANSI TC X3H2

ISO/IEC JTC 1/SC 21/WG 3

Database

Title: (ISO-ANSI Working Draft) Database Language SQL (SQL3)

Author: Jim Melton (Editor)

References:

1) ANSI X3H2-94-079/SOU-003, *(ISO Working Draft) Framework for SQL (SQL/Framework)*, March, 1994

2) ANSI X3H2-94-080/SOU-004, *(ISO-ANSI Working Draft) Database Language SQL*, March, 1994

3) ANSI X3H2-94-081/SOU-005, *(ISO Working Draft) SQL Call-Level Interface (SQL/CLI)*, March, 1994

4) ANSI X3H2-94-082/SOU-006, *(ISO Working Draft) SQL Persistent Stored Modules (SQL/PSM)*, March, 1994

5) ANSI X3H2-94-083/SOU-007, *(ISO Working Draft) SQL Host Language Bindings (SQL /Bindings)*, March, 1994

6) X3H2-94-077, *Minutes from ANSI X3H2 meeting*, February, 1994

7) X3H2-94-185, *Minutes from ANSI X3H2 meeting*, April, 1994

8) X3H2-94-306, *Minutes from ANSI X3H2 meeting*, June, 1994

9) RIO-001, *Minutes from ISO DBL RG meeting*, July, 1994

1 Editorial Notes

The attached SQL3 base document incorporates the changes arising from the ISO DBL RG meeting held January, 1994, in Munich, Germany, and the ANSI X3H2 meeting in February, 1994, in Tiburon, CA.

The following conventions are used in the base document:

1) Words or short phrases that have been approved by ANSI but not by ISO are indicated by:
   ANSI sample phrase
   while words or short phrases approved by ISO and not by ANSI are indicated by:
   ISO example.
   Short differences between ANSI and ISO are indicated by:
   ANSI ANSI version.
   ISO ISO version.

2) Longer sequences of text that have been approved by ANSI, later changed by ISO, and not yet reconsidered by ANSI are distinguished by:

   ANSI Only—caused by ISO changes not yet considered by ANSI

   separating the ANSI-specific text from the preceding text by the convention you see here, and from the following text by the single heavy rule below. This separator is used to identify information that may change when ANSI has the opportunity to consider changes accepted by ISO that affect the area indicated.

   Similarly, longer sequences approved by ISO, changed by ANSI, and not yet reconsidered by ISO are distinguished by:

   ISO Only—caused by ANSI changes not yet considered by ISO

   separating the ISO-specific text from the preceding text by the convention you see here, and from the following text by the single heavy rule below as before. This separator is used to identify information that may change when ISO has the opportunity to consider changes accepted by ANSI that affect the area indicated.

   It is the intent of these conventions to clearly highlight those portions of the SQL3 base document that are not yet approved by both bodies. One hopes that this highlighting will make it easier for the two bodies to move closer, so that a single standard can result from their efforts.

3) I have created a number of additional separators. While recognizing that the proliferation of these separators makes the document somewhat complex to examine, I believe it to be necessary in order that we may know the precise state of each part of the document. In particular, you may find in this document separators as follows:

   ISO Only–SQL3

   This separator indicates that ISO has approved the text following the separator for SQL3. ANSI has not approved the text at all.

ANSI Only–SQL3

This separator indicates that ANSI has approved the text following the separator for SQL3. ISO has not approved the text at all.

4) I am usually able to produce a document that has changebars to indicate changes between the present document and the preceding version. New (inserted) and changed material is marked with a thin vertical bar like that next to this paragraph.

- Example of deletion bullet

  Additionally, places in the current document where material from the preceding version has been deleted are marked with a bullet like the one preceding this paragraph. The note beside the bullet, if any, indicates the number of "entities" that have been deleted. Those entities are typically sentences, although they are sometimes text processor commands.

In Clause 2, "Changes and Notes", I have made a number of notes related to the changes that I have made to Reference 1 to derive this document.

In Clause 3, "Possible problems with the SQL3 language", I have listed a number of possible problems in the SQL3 language. This list should be viewed as a sort of "work item list", helping identify areas that should (must?) be resolved before the document is ready for public review. This material is divided into genuine problems versus those problems that simply *should* be addressed or for which "wordsmithing" is the likely answer; the second category is not viewed as sufficiently urgent to cause the document to fail a ballot for public review or advancement. Finally, there is a new section for "Language Opportunities" that lists those items in which some interest has been expressed, but that are not felt to be sufficiently important to justify delaying the standard in their absence.

In Clause 4, "SQL3 Language Features Not Approved by Both ANSI and ISO", I have listed what I believe to be the differences between ANSI-approved SQL3 features and ISO-approved features at the time of publication of these Editor's Notes.

In Clause 5, "Guidelines for writing "user-friendly" change proposals", I have outlined some suggested guidelines that I encourage you to use when writing change proposals. Following those guidelines will markedly help improve the quality of the document.

2 Changes and Notes

ISO DBL RG changes

In its July, 1994, meeting in Southampton, United Kingdom, the ISO/IEC JTC1/SC21/WG3 Database Languages Rapporteur Group (DBL RG) Meeting considered most of the papers that had been put before it. All those papers that were accepted have been incorporated into this document.

The results of those papers that were approved are reflected in this document; this portion of the Editor's Notes indicates the papers whose changes are reflected.

1) SOU-020/X3H2-93-481, *Privileges: correct definition of "abandoned"*

2) SOU-021/X3H2-93-433, *Large Object Strings*

3) SOU-027/X3H2-93-531R1, *Correction to VIEW_TABLE_USAGE base table*

4) SOU-028/X3H2-93-523R, *Further clarification to COLUMN_DEFAULT*

5) SOU-029/X3H2-93-556, *Null-elimination warning for cursors*

6) SOU-030/X3H2-94-019, *Correction to definition of outer reference*

7) SOU-031/X3H2-94-015, *Casting between datetimes and character strings*

8) SOU-034/X3H2-93-493, *Make column-level SELECT privileges work*

9) SOU-035/X3H2-93-500, *Different types of external routines*

10) SOU-036/X3H2-93-503, *Consistency of descriptor area and diagnostics area*

11) SOU-037/X3H2-93-534, *Fix on Leveling Rule of <referential constraint definition>*

12) SOU-038/X3H2-93-560, *Consistent SQLSTATEs for Referential Integrity constraint violation*

13) SOU-040/X3H2-94-026, *Triggers on views*

14) SOU-041/X3H2-94-027, *Schema paths for data types*

15) SOU-042/X3H2-94-028, *Improving encapsulation I*

16) SOU-045/X3H2-94-031R, *Interval qualifier*

17) SOU-046/X3H2-94-060, *Null class specification*

18) SOU-051/X3H2-94-239, *Common subexpression in queries*

19) SOU-054/X3H2-94-242, *MIscellaneous editorial corrections*

20) SOU-055/X3H2-94-243, *Erratum in Subclause 5.2*

21) SOU-059/X3H2-94-111, *Remove <SQL statement>*

22) SOU-060/X3H2-94-126, *Clean up <query specification>*

23) SOU-062/X3H2-94-158, *PL/I specification of FLOAT and DECIMAL*

24) SOU-063/X3H2-94-102, *Adding functions, routines, overloading to SQL/PSM*

25) SOU-065/X3H2-94-106, *Miscellaneous PSM changes*

26) SOU-069/X3H2-94-148, *Clarify <leave statement> Syntax Rules*

27) SOU-070/X3H2-94-159, *Exception handling in <for statement>*

28) SOU-073/X3H2-94-096, *Changing the type of an <abstract data type value expression>*

29) SOU-074/X3H2-94-098, *Remove semicolon in CREATE TRIGGER*

30) SOU-075/X3H2-94-099, *Fixes to Large Object Strings*

31) SOU-076/X3H2-94-103, *ROW types—base proposal*

32) SOU-077/X3H2-94-150, *Using Row Types in <for statement>*

33) SOU-080/X3H2-94-109, *Restrict use of 'empty' ADT constructor*

34) SOU-081/X3H2-94-178, *Miscellaneous cleanup*

35) SOU-082/X3H2-94-124, *Replace "<column reference>" with "column reference"*

36) SOU-083/X3H2-94-125, *Alignment of ADT descriptor*

37) SOU-084/X3H2-94-130, *Destructors for object ADTs*

38) SOU-085/X3H2-94-131, *Subject routine determination in views/constraints*

39) SOU-086/X3H2-94-143, *SQL-session attributes*

40) SOU-087/X3H2-94-149, *Miscellaneous routine cleanup*

41) SOU-088/X3H2-94-151, *Inner-most atomic execution context*

42) SOU-090/X3H2-94-169R, *Bit strings and <position expression>*

43) SOU-097/X3H2-94-244, *Miscellaneous Possible Problems*

44) SOU-104/X3H2-94-252, *A proposal to fix Possible Problems 261, 298, and 300*

45) SOU-105R1/X3H2-94-253, *Removal of redundant <there is predicate>*

46) SOU-106/X3H2-94-254, *A proposal to fix Possible Problem 262*

47) SOU-107/X3H2-94-255, *A proposal to fix Possible Problem 277 and Language Opportunity 276 (but not Possible Problem 153)*

48) SOU-108/X3H2-94-256, *Fix to Possible Problem 299*

49) SOU-109/X3H2-94-257, *A proposal to fix Possible Problem 313*

50) SOU-112/X3H2-94-260, *Fix to Possible Problem 344*

51) SOU-118/X3H2-94-264, *A proposal to standardize Session Management statement syntax*

52) SOU-130/X3H2-94-275, *Cursor correction*

53) SOU-131/X3H2-94-282, *Ada OUT parameters*

54) SOU-138/X3H2-94-216, *CASE*

55) SOU-139/X3H2-94-289, *Schema information tables for SQL/PSM*

56) SOU-141/X3H2-94-217, *Single-root supertype rule*

57) SOU-142/X3H2-94-231, *Removal of virtual attributes*

58) SOU-145/X3H2-94-233R, *Change "handle" to "locator"*

59) SOU-146/X3H2-94-293/X3H2-94-204, *Improving encapsulation: changes and fixes*

60) SOU-150/X3H2-94-268, *OID Association*

61) SOU-151/X3H2-94-269, *Nullability with INTERSECT and EXCEPT*

62) SOU-154/X3H2-94-284, *Need for <specific type template designator*

63) SOU-184/X3H2-94-201, *Flow of control, exception handling, and transactions*

64) SOU-186/X3H2-94-387, *Response to SOU-071, "Specialize Generalize rows in subtable"*

65) SOU-188/X3H2-94-387, *Comments on Final Text of Corrigendum 1*

66) SOU-???/X3H2-94-116, *ELSEIF usage in the <if statement>*

ANSI X3H2 changes

In its April, 1994, meeting in Pittsburgh, PA, and its June, 1994, meeting in Fargo, ND, the ANSI X3H2 body considered most of the papers that had been put before it in a timely manner.

The results of those papers that were approved are reflected in this document; this portion of the Editor's Notes indicates the papers whose changes are reflected.

1) SOU-020/X3H2-93-481, *Privileges: correct definition of "abandoned"*

2) SOU-021/X3H2-93-433, *Large Object Strings*

3) SOU-027/X3H2-93-531R1, *Correction to VIEW_TABLE_USAGE base table*

4) SOU-028/X3H2-93-523R, *Further clarification to COLUMN_DEFAULT*

5) SOU-029/X3H2-93-556, *Null-elimination warning for cursors*

6) SOU-030/X3H2-94-019, *Correction to definition of outer reference*

7) SOU-031/X3H2-94-015, *Casting between datetimes and character strings*

8) SOU-033/X3H2-93-465, *A Backward-Compatible extension to the FROM clause*

9) SOU-034/X3H2-93-493, *Make column-level SELECT privileges work*

10) SOU-035/X3H2-93-500, *Different types of external routines*

11) SOU-036/X3H2-93-503, *Consistency of descriptor area and diagnostics area*

12) SOU-037/X3H2-93-534, *Fix on Leveling Rule of <referential constraint definition>*

13) SOU-038/X3H2-93-560, *Consistent SQLSTATEs for Referential Integrity constraint violation*

14) SOU-039/X3H2-94-004, *Row identifiers simplify applications*

15) SOU-040/X3H2-94-026, *Triggers on views*

16) SOU-041/X3H2-94-027, *Schema paths for data types*

17) SOU-042/X3H2-94-028, *Improving encapsulation I*

18) SOU-044/X3H2-94-030R1, *Roles module*

19) SOU-045/X3H2-94-031R, *Interval qualifier*

20) SOU-046/X3H2-94-060, *Null class specification*

21) SOU-059/X3H2-94-111, *Remove <SQL statement>*

22) SOU-060/X3H2-94-126, *Clean up <query specification>*

23) SOU-062/X3H2-94-158, *PL/I specification of FLOAT and DECIMAL*

24) SOU-063/X3H2-94-102, *Adding functions, routines, overloading to SQL/PSM*

25) SOU-065/X3H2-94-106, *Miscellaneous PSM changes*

26) SOU-069/X3H2-94-148, *Clarify <leave statement> Syntax Rules*

27) SOU-070/X3H2-94-159, *Exception handling in <for statement>*

28) SOU-072/X3H2-94-092, *Specifying triggers*

29) SOU-073/X3H2-94-096, *Changing the type of an <abstract data type value expression>*

30) SOU-074/X3H2-94-098, *Remove semicolon in CREATE TRIGGER*

31) SOU-075/X3H2-94-099, *Fixes to Large Object Strings*

32) SOU-076/X3H2-94-103, *ROW types—base proposal*

33) SOU-077/X3H2-94-150, *Using Row Types in <for statement>*

34) SOU-080/X3H2-94-109, *Restrict use of 'empty' ADT constructor*

35) SOU-081/X3H2-94-178, *Miscellaneous cleanup*

36) SOU-082/X3H2-94-124, *Replace "<column reference>" with "column reference"*

37) SOU-083/X3H2-94-125, *Alignment of ADT descriptor*

38) SOU-084/X3H2-94-130, *Destructors for object ADTs*

39) SOU-085/X3H2-94-131, *Subject routine determination in views/constraints*

40) SOU-086/X3H2-94-143, *SQL-session attributes*

41) SOU-087/X3H2-94-149, *Miscellaneous routine cleanup*

42) SOU-088/X3H2-94-151, *Inner-most atomic execution context*

43) SOU-089/X3H2-94-162, *Default Roles*

44) SOU-090/X3H2-94-169R, *Bit strings and <position expression>*

45) SOU-130/X3H2-94-275, *Cursor correction*

46) SOU-131/X3H2-94-282, *Ada OUT parameters*

47) SOU-138/X3H2-94-216, *CASE*

48) SOU-139/X3H2-94-289, *Schema information tables for SQL/PSM*

49) SOU-141/X3H2-94-217, *Single-root supertype rule*

50) SOU-142/X3H2-94-231, *Removal of virtual attributes*

51) SOU-145/X3H2-94-233R, *Change "handle" to "locator"*

52) SOU-146/X3H2-94-293/X3H2-94-204, *Improving encapsulation: changes and fixes*

53) SOU-150/X3H2-94-268, *OID Association*

54) SOU-151/X3H2-94-269, *Nullability with INTERSECT and EXCEPT*

55) SOU-154/X3H2-94-284, *Need for <specific type template designator*

56) SOU-???/X3H2-94-116, *ELSEIF usage in the <if statement>*

Other changes

In addition, the following changes appear in this document.

1) A number of minor editorial changes intended to clarify vague wording, incorrect grammar, inconsistent phraseology, and so forth.

3 Possible problems with the SQL3 language

I observe some possible problems with SQL3 as defined in this document. These are noted below. Further contributions to this list are welcome. Deletions from the list (resulting from change proposals that correct the problems or from research indicating that the problems do not, in fact, exist) are even more welcome. Other comments may appear in the same list.

Because of the increasingly dynamic nature of this list (problems being removed because they are solved, new problems being added), it has become rather confusing to have the problem numbers automatically assigned by the document production facility. In order to reduce this confusion, I have instead assigned "fixed" numbers to each possible problem. These numbers will not change from printing to printing, but will instead develop "gaps" between numbers as problems are solved.

Possible problems that may be SQL-92 Errata and that relate to SQL3

Significant Possible Problems:

999 In the body of the Working Draft, I have occasionally highlighted a point that requires urgent attention thus:

****Editor's Note****
Text of the problem.

These items are indexed under "**Editor's Note**".

Minor Problems and Wordsmithing Candidates:

091 The Description section for some SITs are redundant. For example, the COLUMN_PRIVILEGES table clearly specifies the content (1 row per privilege descriptor). But the Description section also attempts to specify the content, this time in a more procedural manner. The redundancy introduces the possibility of inconsistencies. For example, the Description section of COLUMN_PRIVILEGES table fails to consider the case where privilege descriptors are added (e.g., see General Rules of Subclause 11.15, "<add column definition>"). (Source: Jon Bauer)

LON-133/X3H2-90-515 mentioned:

> In addition, the document seems to rule out times such as TIMESTAMP '1989-12-31:23:59:60' (i.e., representation of a "leap second" value). However, what happens when a <datetime value function> such as CURRENT_TIMESTAMP should return such a time is not discusses. Should the implementation wait for the leap second to pass by, are non-monotonic timestamps allowed, or should "...59" be returned for two consecutive seconds?

155 This Possible Problem has been moved and has now become Possible Problem **PSM-020**.

155 SEL-84 suggests two stylistic issues that "ought to be resolved":

i) There are several methods used to refer to the prepared statement – see existing GRs 2, 3, and 4, for example [I believe that the Rules in question are in Subclause 13.10, "<execute statement>"]. There probably should be a GR of the form "Let PS denote..." and then use PS to refer to the prepared statement thereafter.

ii) We also need to be consistent regarding whether we use <dynamic select statement>/<dynamic single row select statement> or <cursor specification>/<query specification>. We should surely use the former when referring to a prepared statement.

167 Ed Dee has noted with regards to connections:

> Dynamic SQL: This is not an area in which I am at all strong but I would have thought that a prepared statement would now not only have to be bound to a module but also to an SQL user session. Otherwise, it would be valid for a statement to be prepared in one SQL user session and executed etc in another and that seems rather liberal to me. The clauses I believe require attention are:
>
> — As an Editorial aside, [Ed notes] that SR 1 and GR 3 (of Subclause 13.10, "<execute statement>"?) appear to conflict in concept. [Ed *guesses* that GR 3 should be trying to tie the temporary table definitions into the same module as the prepare/execute, but that is not what it says.

179 Pedro Celis submitted the following Possible Problem in his response to TC LB X3H2-90-266:

> Dynamic SQL. You can reference a statement prepared in another module (if you specify the statement name with a host variable), but you can't reference a cursor declared in another module. The rules for cursors should match the rules for statements.

195 Jim Melton, in his response to TC LB X3H2-90-267, said:

> The DATA parameter in the dynamic system descriptor area causes a problem: what should the data type of the *parameter* be that is used to retrieve or store the DATA value? SQL2 currently specifies that DATA has a datatype appropriate to TYPE, LENGTH, etc. However, this begs the question of what a <parameter specification> would have to specify in order to retrieve (or set) the value of DATA. Solutions exist, of course, but they are awkward.

205 Jonathan Bauer has noted the following minor problem (which I paraphrase):

The General Rules of many Subclauses, such as Subclause 11.5, "<table definition>", don't actually say that the object (table, in this example) is actually created! In this example, General Rule 3) says that the table *descriptor* is created, but says nothing about the table itself. This is not a problem when a <table definition> is contained in a <schema definition> because General Rule 2) of Subclause 11.1, "<schema definition>", says that "a schema is created that...". However, when the <table definition> is used as an <SQL statement> in a <procedure> in a <module> (but not in a <schema definition>), there is no Rules that causes the table itself to be created.

No doubt, the solution is to find all such Subclauses in Clause 11, "Schema definition and manipulation", and revise the GRs to say something like "...if [this statement] is executed as an <SQl statement> in a <procedure> in a <module>, then the [appropriate object] is created; otherwise (in a <schema definition>), the table is 'tentatively created' pending creation of the schema itself" or words to that general effect.

215 While merging several papers from the SQL-92 second CD draft Editing Meeting that related to the General Rules of Subclause 13.9, "<using clause>", the Editor noticed the following problem with those General Rules:

General Rules 8) and 9) (in X3H2-90-398/LON-003) are generally structured like this:

8) When a <using clause> is used in *something or other* . . .

   Let *D* be . . .

   Case:

   a) If <using arguments> is specified and the number of <argument>s is not *D*, then an exception condition is raised: *dynamic SQL error— using clause does not match dynamic parameter specifications*.

   b) If <using descriptor> is specified, then:

      i) *Many rules dealing with TYPE, LENGTH, NAME, PRECISION, etc.*

   c) If <using descriptor> is specified, then:

      i) *Some rules dealing with automatic CASTs*

However, that structure is, at the least, somewhat confusing. It is far from clear whether the Rules dealing with TYPE, etc., are specifying what the *database system* will cause to happen, or what the required behavior of the application program is.

The structure of the cited Rules should be changed to make it quite clear which of these alternative interpretations is intended. The Rules are currently worded (and were already so worded even before LON-139/X3H2-91-034 and other papers were applied) in a way that suggests that the behavior of the database system that an application program can depend on is being described; however, research and experimentation indicates that the Rules really describe required behavior of the application program!

218 Ken Jacobs has noted:

We have now addressed in X3H2-91-040 the need for an implementation to add new tables and views to the Information Schema and to add new columns to existing tables and views in the Information Schema. We have yet to address the vendor requirement to add new rows, such as might be desired if a vendor creates a new base data type (money, for example), for example. We need to propose some way to permit new rows of the Information Schema to have values that

are not permitted by the existing constraints in the current Information Schema definitions. Note that the Note in Subclause 18.1, "Introduction", indicates that "Constraints defined in this Clause are not actual database constraints.

252 Hugh Darwen has noted:

In Subclause 11.34, "<drop domain statement>", General Rule 2) says that *TN* can be the name of a <temporary table declaration>, but "ALTER TABLE *TN* ADD *TCD*" doesn't seem reasonable. Perhaps we should eliminate <drop action> from <drop domain>?

316 The Munich amendment to MUN-047/X3H2-93-238, as codified in MUN-191, noted the following problem:

The term "function invocation" is widely used in the SQL3 document and should probably be replaced by something like "routine invocation, where the subject routine is a function" or similar wording.

402 Somebody (unidentified) noted in Southampton:

There is a surplus semicolon in Annex G in the Functions (See SOU-112 and SOU-087).

403 Paper X3H2-94-259/SOU-111 noted the following Possible Problem:

The General Rules in Subclause 13.2, "<open statement>", deal with CASCADE ON and OFF and depend on the constraint-referential-action graph, which has been deleted. Therefore, these rules will need to be revised.

Possible problems related only to SQL3

005 Clause 14, "Embedded SQL", Syntax Rule 9)d), says that there is "...one <procedure> PS that contains an <open statement> that references..." each cursor. However, we now have options on <open statement>: paper SYD-117/X3H2-89-052 added an optional READ-ONLY/UPDATE clause. That now means that a single <procedure> for all <open statement>s in a <module> is not adequate–there are multiple flavors of <open statement> possible (with FOR READ ONLY, with FOR UPDATE without a column list, and all the variations of column lists). Paper X3H2-89-131 proposed permitting multiple <procedure>s in a <module> to contain an <open statement> for a specific cursor. This paper was rejected because of problems involving the order and datatype restrictions on the parameters to those <procedure>s. The original problem still exists. See also **079**.

011 Paper X3H2-89-087 has unsuccessfully addressed a problem with a permitted use of CURRENT, CURRENT_UTC, USER, and SYSTEM USER. Interactions between these functions and referential actions, and between these functions and nested views with WITH CHECK OPTION, must be resolved. Paper FIR-119/X3H2-89-367 fixed this for SQL2, but did not address all aspects of SQL3.

032 In Subclause 11.45, "<trigger definition>", the relationship between R and SU is not clear. R is defined in association with S in GR1, but the relationship between R and SU is not defined.

033 Conventions for DBMS-initiated ROLLBACK (e.g., for deadlock): in synchronous mode, the statement following a DBMS-initiated rollback would begin a new transaction. In asynchronous mode, statements following a DBMS-initiated rollback should be rejected until a user-initiated rollback or commit is executed.

034 The TEST COMPLETION statement returns output parameter or host variable parameters for the statement(s) that completed. The manner by which these parameter "addresses" are saved by the DBMS is difficult to define, and such "saved parameter addresses" are a new sort of programming problem for the user.

035 Paper X3H2-88-341 (a/k/a SYD-55) identified a parsing problem that has been introduced into SQL3.

In Subclause 7.15, "<recursive union>", both the optional <recursive column list> and <iteration expression> start with a parenthesis, requiring a look-ahead of greater than one. It is not sufficient to move the <correlation name list> and <recursive column list> to follow the <iteration expression> since this also causes parsing problems (e.g., an identifier at the end of the <iteration expression> followed by a table name can look like a parameter followed by an indicator parameter). One solution is to give a keyword to the iteration expression. Another solution is to move the <correlation name> and <recursive column list> to follow the right parenthesis of the <recursive union> as was originally proposed. Still another solution is to remove some of the flexibility and require a <recursive column list>.

036 Paper X3H2-88-341 (a/k/a SYD-55) identified a parsing problem that has been introduced into SQL3.

In Subclause 13.8, "<insert statement>", the INSERT INTO <table reference> syntax has a lookahead problem. The optional <derived column list> for a <table name> or <derived table> in a <table reference> starts with an open parenthesis. Furthermore, the optional <insert column list> (and perhaps the <query expression>) of the <insert statement> also starts with a parenthesis. To disambiguate this requires a lookahead of greater than 1. This feature is probably of marginal utility (the authors of the original proposal stated this), and the easiest solution would be to simply restore <insert statement> to require a <table name> instead of a <table reference>.

038 Paper X3H2-88-341 (a/k/a SYD-55) identified a parsing problem that has been introduced into SQL3.

In Subclause 6.19, "<enumerated value expression>", there is a lookahead problem. <enumerated value expression> can go to "<domain name> (<value expression>)".

Minimally, there must be a rule on <external procedure declaration> that states that the <procedure name> must be different from any <domain name>, caused by a conflict in <value expression> that can include a "<domain name>(<value expression>)" or a "<procedure name>(<value expression>)". Assuming the rule is added, this conflict can be finessed in the parser.

However, the appearance of <external function invocation> in <boolean expression> is more difficult to handle. It can be argued that, rather than supporting boolean procedures, the procedure should return 0 or 1 (as in General Rule 2) and the application test for those values as TRUE or FALSE (e.g., "where func()" would become "where func() = 1"). It would also be possible for the procedure to return a domain defined with "TRUE" and "FALSE". This would mean removing " | <external function invocation>" from <boolean primary> and GR2.

039 The CASCADE OFF cursor mode ("set processing mode") raises several problems: 1) It introduces a limited subtransaction concept, under the covers, if the goal of behaving like a single searched update/delete statement is accepted (an atomic group of statements is created); 2) This is further complicated by recent generalizations that allow unrelated SQL statements to occur during set processing mode–this effectively creates an interleaved subtransaction; 3) Though 1) specifies the intent of this construct, in fact, no provision for rollback to the state prior to initiation of set processing mode is made–in fact, any rollback is prohibited allowing a user to be 'trapped'.

066 Paper X3H2-89-314/FIR-92 suggests that General Rules in Subclause 11.20, "<drop column definition>" that are intended to handle dropping subtables might not actually work and produce the desired result.

075 Paper FIR-78/X3H2-89-329) discussed possible problems with certain join shorthands. The problems primarily deal with the lack of key inheritance by <query expression>s, <joined table>s, etc. Furthermore, the paper points out a certain vagueness in places like the syntax rules of Subclause 7.9, "<joined table>" where the term "identified by" is used without adequate definition.

079 In order to keep the one to one relationship between open statements and cursor declarations it will be necessary to allow the parameterisation of the CASCADE ON | OFF options on the open statement (e.g., CASCADE :x). (Ref: Firenze minutes section 17.1) (Source: Phil Shaw) See also **005**.

080 FIR-110, which fixes privileges (especially the requirements for SELECT privileges), does not address SQL3 issues, in particular Triggers, Assertions, and the use of <table reference> instead of <table name> in several DML statements. (Ref: FIR-110) (Source: Ed Dee)

082 Some data types are implicitly "CAST" into others (e.g. NATIONAL CHARACTER into CHARACTER) but not all (e.g. DATETIME). Also this implicit casting works only for the module language and not for the embedded languages. This is inconsistent and there is therefore an opportunity for improvement. (Source: Phil Shaw)

086 The addition of a standard error return mechanism to external procedures would be a useful enhancement (Source: Phil Shaw)

087 The mappings between SQL data types and host language datatypes are currently presented in sequences of text rules. The specification would probably be more concise and more easily understood if the mappings were presented in tabular form. (This table could then be referenced from the many places where the current rules are replicated.) (Source: Phil Brown)

153 Paper SEL-106 says

> The General Rules for updating, deleting, and inserting rows in <joined table>s using the <delete statement: positioned>, <delete statement: searched>, <insert statement>, <update statement: positioned>, and <update statement: searched> are incomplete and inadequate. The

General Rules are all based on a "single" affected row, but in the case of a <joined table>, more that one table might be updated For example, consider General Rule 10) of Subclause 13.9, "<update statement: positioned>". This Rule clearly states "The object row", i.e., a singular row. In addition, there are other inconsistencies. For example, consider Syntax Rule 3) of Subclause 13.9, "<update statement: positioned>". This rule specifies the need for an INSERT privilege, but no inserts can result from the existing General Rules.

216 Phil Shaw and Steve Cannan have noted:

A DROP DOMAIN that specifies a domain with an enumerated data type cannot be defined as it currently is, because the data type cannot be propagated to the column definition as it is for other data types.

218 Bruce Horowitz and Keith Hare have both noted:

It is not specified what happens when a referential action of SET NULL (<null state>) is involved in a referential constraint. The GRs for <referential constraint definition> only talk of seting to the general null value. This should be explored.

Furthermore, the multiple nulls should be better integrated with the enumerated data type (at least so that the definitions are analogous).

Finally, there probably needs to be a specification in <null predicate> to ensure that it's possible to see if a value is "some null value", a specific null value, or even one of a list of null values.

223 Ed Dee has noted:

X3H2-90-166/LON-012 added text that references <table name>, but SQL3 has <table reference> in the update/delete statements. The text must be corrected for SQL3.

245 Jonathan Bauer pointed out in X3H2-91-093:

For some reason, you can specify ALL PRIVILEGES for all objects except external procedures and data types. This means, for instance, that you can enter

REVOKE ALL PRIVILEGES ON DOMAIN D but not

REVOKE ALL PRIVILEGES ON DATA TYPE D [Jonathan assumes] that we will want to clean this up at some point.

246 Discussion of ARL-012 and ARL-048 pointed out that the correlation between Ada library names and qualified module names might be a problem. Furthermore, these two papers did not add Information Schema tables and views to reflect the new schema objects, but they should have.

247 ARL-050 posed the following problem:

Paper ARL-050 (Hugh Darwen) describes some problems concerning updating query expressions that use UNION and may deliver duplicate rows: the same row can be deleted more than once and an updated current row of a cursor has an ambiguous mapping to the database.

248 ARL-051 posed the following problem:

Paper ARL-051 (Hugh Darwen) describes some problems concerning updating query expressions that use JOIN USING FOREIGN KEY. The General Rules for INSERTing into such a join are not sufficiently careful about their use of the term "matching row" and a case is noted that, where the left operand of the JOIN permits duplicate rows, an insert of a single row can result in several rows being inserted into the query result.

249 In X3H2-91-274R2 Jonathan Bauer identified the following problem:

> [Jonathan believes that] an <abstract data type definition> is abandoned if any privilege required to create it is no longer present. For instance, an ADT that has a method that updates a table T should be abandoned if UPDATE on T is lost. [Jonathan thinks that] SR 20 should be reworded to match the wording of SR 19 and SR 21.

250 In KAW-055/X3H2-92-030 JPN-8, Japan identified the following problem:

> The General Rules of Subclause 13.8, "<insert statement>", in SQL-92 have been written in terms of "leaf generally underlying table" (LGUT) because updatable views in SQL-92 can have only one LGUT. However, updatable views in SQL3 can have more than one LGUT; consequently, the GRs of Subclause 13.8, "<insert statement>", have to be revised to take this into account!

251 In KAW-058/X3H2-91-256 USA#45.4, the following possible problem/opportunity was identified:

> The owner of a view can hold the UPDATE and/or INSERT table-level privileges on that view; such privileges are meaningless. Since no column can be added to a view, the fact that he/she holds this privilege is never acted upon. See also Language Opportunity **244**.

253 In CBR-016/X3H2-92-191, the following possible problem was identified:

> X3H2-92-084/OTT-010, section 1.5, "<routine invocation>", Syntax Rule 4) evokes the question: We don't fully understand about "the catalog of the <routine invocation>". Are all <routine invocation>s now in catalogs? There was no corresponding rule for <external function invocation>s.

In X3H2-92-248/CBR-045, part 3.13, the USA responded:

> The proposal copied the phrase about "the catalog of the <routine invocation>" from the prior specifications, specifically Syntax Rule 1) ("The <SQL function name>...") of Subclause 10.5, "<SQL function invocation>". The phrase should probably specify "the catalog of the module or schema that contains the <routine invocation>", but it isn't clear whether a <module> actually is in a <catalog>.

254 In CBR-017/X3H2-92-192, the following possible problem was identified:

> X3H2-92-083/OTT-011, Syntax Rule 4), regarding compatibility of the source and target data types seems only to cater for assignments to ADTs in its second half. This needs rectifying.

255 In CBR-017/X3H2-92-192, the following possible problem was identified:

> Following back the rules for <template parameter name>, we come to Syntax Rule 22) of Subclause 5.4, "Names and identifiers". Part b) of this Rule states that "A <template parameter name> that is *contained* in a <value specification> shall be *contained* in a <template parameter declaration>..." We assume that this is not intended to mean that the <value specification> is contained in the <template parameter declaration>, so we think this Rule needs rewording.

256 In CBR-017/X3H2-92-192, the following possible problem was identified:

> There is no General Rule in Subclause 5.4, "Names and identifiers", that states what a <template parameter name> is or does. This is incompatible with the rest of the Subclause.

257 In CBR-017/X3H2-92-192, the following possible problem was identified:

> The interactions between <assignment statement> and Subclause 4.36, "SQL-statements", is very incomplete. Given the <assignment statement>'s new general role, we believe that its classification as an "SQL-control statement" (Subclause 4.36, "SQL-statements") is counter-intuitive but perhaps there is nowhere else to put it. More serious, however: as the result of the proposal

in X3H2-92-083/OTT-011, an <assignment statement> can now contain a <dynamic parameter specification>. We guess that it should therefore be listed as a "preparable statement" in Subclause 4.36, "SQL-statements" (and perhaps other "SQL control statements" listed as not preparable?). Is an <assignment statement> immediately executable? Is it embeddable? These points also need addressing for other "SQL-control statements".

258 In CBR-017/X3H2-92-192, the following possible problem was identified:

On the assumption that <assignment statement> is preparable, shouldn't there be some subrule in Subclause 13.6, "<prepare statement>" (akin to General Rules 4 and/or 5) to describe the effect (or allowability?) of cases such as SET ? = ?

264 Consideration of X3H2-92-???/CBR-039 identified the following Possible Problem:

If a column is added to a supertable, it is not clear what happens if one or more of its subtables already has a column of that name. Note that, unlike the case where a subtable is created, there is no way to give the inherited column a different name from its name in the supertable. See General Rule 6) of Subclause 11.15, "<add column definition>".

In Subclause 11.20, "<drop column definition>", General Rule 3) says:

3) Of *T* is a supertable, then all columns inherited from *CN* in any subtable of *T* are also destroyed.

General Rule 4) is evidently intended to have the same effect, but uses the *CN* as a <column name> for the subtable, whereas it may have been changed. General Rule 4) should therefore be deleted.

265 Consideration of X3H2-92-???/CBR-039 identified the following Possible Problem:

In Subclause 11.21, "<add supertable clause>", General Rule 2) adds descriptors for newly inherited columns to the descriptor of *T*, with possibly replaced <column name>s.

In General Rule 3), by performing an <add column definition> for each such column, the same is done again, but without the possibility of replacing the names. However, this time (presumably) the columns are added, as well as their descriptors.

266 Consideration of X3H2-92-???/CBR-039 identified the following Possible Problem:

In Subclause 11.22, "<drop supertable clause>", the Access Rules should be General Rules.

Access Rule 2) requires columns to be identified that are inherited form a specific supertable. A column descriptor contains no indication of whether the column is inherited and, if so, from what column of what supertable. See also Possible Problem **067**.

267 Consideration of X3H2-92-???/CBR-039 identified the following Possible Problem:

In Subclause 18.4.30, "COLLATIONS base table", there is no indication of whether the collation is actually a translation followed by a collation as specified in Subclause 11.39, "<collation definition>", General Rule 6). Shouldn't there be?

269 Consideration of X3H2-93-015/YOK-025 identified the following Possible Problem:

In Subclause 11.45, "<trigger definition>", a new Syntax Rule was added by the paper. That Rule makes it permissible to define a trigger on a <query expression>, which defines an unnamed virtual table. However, succeeding Rules require that the table on which the trigger is defined have a name.

270 Consideration of X3H2-93-049/YOK-028 identified the following Possible Problem:

Throughout the document, we have Rules that require the implementation to "invent" an implementation-dependent name for something. However, those Rules don't specify what the catalog and schema components of that name should be. Should the name be in the same catalog and schema as the object currently being processed? Is that a meaningful concept in general? Do some of those names have "predefined" components? For some of them, are the catalog and schema components meaningless?

271 Consideration of X3H2-93-039/CBR-033 identified the following Possible Problem:

The use of Functional Dependencies in GROUP BY may have a problem when certain schema manipulation operations take place.

Consider the following table and view definition:

```
CREATE TABLE t1
   ( dnum ...,
     dname ...,
     location ...,
     CONSTRAINT pk PRIMARY KEY (dnum)
   ) ;

CREATE VIEW v1 AS
   SELECT    dnum, dname, location
   FROM      dept
   GROUP BY  dnum ;
```

Now consider two possible DDL statements:

1) ALTER TABLE t1 DROP COLUMN location CASCADE ;

2) DROP CONSTRAINT pk CASCADE ;

In the first case *v1* is dropped. In the second case the view is no longer valid, but no rules exist to drop it.

272 Consideration of X3H2-93-066/YOK-035 identified the following Possible Problem:

The Format for subtable contains rules for function renaming, and yet the Syntax Rules only discuss inheriting *columns}* rom supertables. It appears that there need to be rules added to discuss inheriting functions.

273 Consideration of X3H2-93-066/YOK-035 identified the following Possible Problem:

Despite having syntax for column and function renaming, there appear to be inadequate (or no) SRs or GRs to talk about how the renaming impacts table and routine descriptors.

274 Consideration of X3H2-93-066/YOK-035 identified the following Possible Problem:

Syntax Rule 6 of Subclause 11.14, "<alter table statement>", ("If *T* has a descriptor containing the name of an abstract data type *ADT* . . . ") prevents *any* table from having a column added, dropped, or altered or a supertable added or dropped, since *all* tables are currently defined to have an abstract data type associated with them.

278 Consideration of X3H2-93-066/YOK-035 identified the following Possible Problem:

In Subclause 13.7, "<delete statement: searched>", the General Rules invoke a destructor function, but when the destructor function for a type is invoked, there is no guarantee that a <destroy statement> will actually be executed and the object ADT instance destroyed. corrected.

279 This Possible Problem has been moved and is now Possible Problem **SOU-021**.

282 Paper YOK-046R1/X3H2-93-185 identified the following Possible Problem:

The Definition and Information Schemas do not have representations for the privileges "ALL SCHEMA PRIVILEGES", "UNDER", or "EXECUTE", which are all defined in Subclause 10.4, "<privileges>". See Subclause 18.1, "Introduction".

283 Paper YOK-051/X3H2-93-190 identified the following Possible Problem:

Subclause 4.30, "Routines", does not adequately describe the concepts of dynamic binding and subject function selection.

284 Paper YOK-051/X3H2-93-190 identified the following Possible Problem:

Clause 4, "Concepts", does not adequately describe the Exception Handling mechanism. The only text currently is in Subclause 4.32.1, "Status parameters", and this may not be the right place.

285 Paper YOK-051/X3H2-93-190 identified the following Possible Problem:

The concept and use of the <operator name list> (OPERATORS <operator name>...) is not described in Clause 4, "Concepts", and in Subclause 11.48, "<abstract data type body>", where the syntax production is there are no General Rules and only one trivial Syntax Rule. Furthermore the operator name list does not currently form part of the abstract data type descriptor (see Subclause 4.11, "Abstract data types".

286 Paper YOK-051/X3H2-93-190 identified the following Possible Problem:

In Subclause 8.15, "<distinct predicate>", General Rule 4) ("The Syntax and General ..."), because it explicitly does not enforce Access Rules, allows objects to be created without having USAGE on the *ADTN* used (or alternatively the privileges needed to create an equivalent ADT to *ADTN* oneself. Furthermore, General Rule 7) ("If the applicable ...") is redundant given General Rule 6) ("A privilege descriptor ...)".

287 Paper YOK-051/X3H2-93-190 identified the following Possible Problem:

The scope of an <exception name> is not defined but is used in Subclause 14.14, "<signal statement>", and in Subclause 14.15, "<resignal statement>".

288 Paper YOK-051/X3H2-93-190 identified the following Possible Problem:

In Subclause 14.6, "<exception declaration>", Syntax Rule 2) refers to the rules of Subclause 19.1, "SQLSTATE". It is not clear if this refers to the general division of the codes into implementation defined and standard defined groups or to the table of standard codes. If the intension is to allow the user to define new application specific codes then the code space needs to be reallocated into implementation, standard and user defined areas.

289 Paper YOK-051/X3H2-93-190 identified the following Possible Problem:

The document is inconsistent with regard to the creation of descriptors for temporary objects. Cursors and declared temporary tables do not have descriptors but temporary views and routines that are local do.

290 Paper YOK-035/X3H2-93-066 identified the following Possible Problem:

When mapping an <embedded SQL host program> to an implied <module> and an implied host program, does the *order* of the declarations (including cursor declarations) need to be preserved?

292 Paper YOK-124/X3H2-93-214 identified the following Possible Problem:

In Subclause 11.21, "<add supertable clause>", General Rule 2), the descriptor of *T* must contain the descriptor of *TG*, nor just the descriptors of the columns of *TG*—otherwise, <drop supertable clause> won't work. The rule needs some rewriting.

293 Paper YOK-124/X3H2-93-214 identified the following Possible Problem:

In Subclause 11.52, "<drop data type statement>", General Rule 5), implementations are required to drop all modules that references the data type being dropped. However, if the <drop data type statement> itself is being executed from a stored module, then presumably it must be dropped itself. The document makes no statement about when that drop becomes effective (*e.g.*, upon COMMIT, when the SQL-agent terminates, upon DISCONNECT). Possibly an approach that better defines what is meant by "references" would be helpful. THe paper also points out that similar problems exist for revoking privileges on modules that are currently executing.

294 Discussion in Yokohama on paper YOK-142/X3H2-93-253 identified the following Possible Problem:

In Yokohama, a basic problem was noted with this Subclause. The BNF non-terminal <user-defined operator symbol>, referenced in the last General Rule of this Subclause, "exists" only when an application program is being compiled . . . not when the <operator definition> is being executed. Now, of course, we mean that the definition of the operator is "effectively" added to the alternatives for <user-defined operator symbol>, but some people still believe that the approach taken in the General Rules of this Subclause needs to be carefully thought out.

295 Discussion in Yokohama on paper X3H2-93-281/MUN-??? identified the following Possible Problem:

Subclause 11.21, "<add supertable clause>", General Rules 2) and 3) state:

2) The column descriptors of *TG* that are not already inherited by *T* are appended to the descriptor of *T* as inherited columns . . .

3) For every newly-inherited column in *T*, an <add column definition> is effectively performed for that column without further Access Rule checking.

ISSUE: What attributes of the inherited column are specified in the <add column definition>? Do the inherited attributes include only those stored in the column descritpor? Or are all <table element>s associated with the inherited columns also inherited?

— Background Point 1: When a table *T* is created as a subtable of an existing table, Subclause 11.5, "<table definition>", General Rule 5)b) ("Every <table element> of every supertable is inherited by the subtable *T*.") means that all column definitions, routine declarations, table constraints, etc. are inherited by *T*.

— Background Point 2: Subclause 11.5, "<table definition>", Syntax Rule 7) ("If LIKE is specified . . . ") has a Note that points out that <column constraint>s are not really included in <column definition>s because such <column constraint>s are "effectively transofrmed to <table constraint>s and are thereby excluded". Or, put another way, neither column nor table constraints are part of a column descriptor, so, are they inherited?

296 Discussion in Yokohama on paper X3H2-93-281/MUN-??? identified the following Possible Problem:

> Numerous access rules state something similar to Subclause 11.6, "<column definition>", AR 3):
>
> 3) If <null class name> is specified, then the applicable privileges shall include USAGE.
>
> The problem is that the above access rule does not specify what object USAGE is needed against.

297 Someone at the Yokohama DBL RG meeting provided the Editor with the following Possible Problem:

> It is not clear whether errors (exception conditions) are stacked onto the diagnostics area or replace the existing codes (except for SIGNAL and RESIGNAL, which are explicit).

299 Steve Cannan noted in Yokohama:

> Should we not have a Conformance Clause in the SQL/PSM and SQL/CLI documents (if they are Parts)?

The Editor has inserted a Conformance Clause into those two documents (as well as SQL/Bindings), but requires that the Committees examine them and make proposals to correct them.

301 Steve Cannan noted in Yokohama:

> YOK/069/X3H2-93-069 used the phrase "innermost atomic execution context" in Subclause 14.8, "<rollback statement>", but does not define what is meant by *nesting* of atomic execution contexts.

302 Steve Cannan noted in Yokohama:

> The rules for COMMIT implicitly execute a SET CONSTRAINTS MODE statement. However, in an implementation that supports multi-session transactions, this SET CONSTRAINTS MODE statement needs to be "executed" for every session that has participated in the transaction.
>
> This is also a problem in SQL-92, so an Errata is in order.

303 Steve Cannan noted in Yokohama:

> The SET TRANSACTION statement is executed on *the* SQL-server (see Subclause 4.42, "Client-server operation"). However, in an implementation that allows multi-session transactions, this statement needs to be executed on *every* SQL-server that participates or will participate in the transaction.

304 Steve Cannan noted in Yokohama:

> Subclause 4.20, "Tables", defines a subtable family based on the subtable tree under a maximal supertable (that is, using a maximal supertable as the starting point). It refers to *a* subtable family of a table *T* (subtable or supertable) and not *the* subtable family. However, Syntax Rule 25) (in X3H2-93-159/YOK-003), reading "If a <table definition> *TD* specifies a <subtable clause>, then:", subrule e), reads "There shall be exactly one maximal supertable in *the* subtable family to which the *supertable* belongs." (emphasis added).
>
> [Steve] can find no other rule that ensures that a subtype [sic] belongs to only one subtable family and so this rule makes an unsubstantiated assumption on the number of subtable families present.

305 Steve Cannan noted in Yokohama:

The name of the TRIGGERED_COLUMNS base table (Subclause 18.4.34, "TRIGGERED_ COLUMNS base table") is confusing and conflicts with the names of the columns of the base table. A clearer description of the function is desired together with a consistent set of names.

306 Steve Cannan noted in Yokohama:

The Information Schema information is missing for routines (a skeleton is present, but needs to be filled in). See X3H2-93-050/YOK-029.

307 David Beech points out that consideration of X3H2-93-112/YOK-089 identified the following Possible Problem: :

In Subclause 11.48, "<abstract data type body>", private attributes are inherited as part of the representation of an instance (as they must be), but they cannot be renamed, since that would violate encapsulation. Syntax Rule 15) states that "in *DTR* [the effective <member list>], no <column name> shall appear more than once". Renaming should not be required to resolve conflicts with inherited private names, since the inherited private name is not validly referenceable in the subtype and could safely be ignored; and if two private attributes of the same name are inherited by multiple inheritance, it is not then possible to rename either of them.

A possible solution is to relax SR 15 to exclude private inherited attributes, since this has no effect on the language itself. Also SR 29 "The descriptor of the abstract data type ADT..." probably intends to put into the descriptor only the non-inherited attributes and routines, to avoid redundancy in the schema and consequent complications in describind schema manipulation. It should use the new "explict" terminology to avoid confusion with the "implicit" and "effective" <member list>s mentiones in Syntax Rules 11 and 14.

Finally, Subclause 6.5, "<component reference>", also needs some tightening up in how it refers to an attribute within an instance. This should be done together with the solution of Possible Problem **273** on renaming.

308 David Yach noted the following Possible Problem in X3H2-93-091:

In Subclause 11.52, "<drop data type statement>", dropping a subtype does not ensure that all instances of values of the subtype are dropped. If a table *T* has a column *C* of some type *ADT1*, some values in *C* might be of type *ADT2*, where *ADT2* is a subtype of *ADT1*. If the type *ADT2* is dropped, the rules do not state what happens to the instances of type *ADT2* in *C*.

310 Jeff Fried noted the following Possible Problem in X3H2-93-428:

Syntax Rule 4) of Subclause 11.54, "<grant statement>", states:

4) If one or more <grantee>s is a role, then the WITH GRANT OPTION <privileges> containing the <action>s REFERENCES, UNDER, or shall not be specified.

However, Subclause 11.56, "<grant role statement>", makes it permissible to grant a role to another role WITH ADMIN OPTION. Now, if some role A is to granted role B WITH ADMIN OPTION, then any <authorization identifier> can grant role A provided role B is granted to C. This violates Access Rule 1) of the <grant role statement>, which is

1) The applicable <authorization identifier> shall have all roles identified by the specified <role granted>s as applicable roles with the WITH ADMIN OPTION.

Also, there is no rule that says who the grantor should be in the role authorization descriptor.

312 Hugh Darwen noted the following Possible Problem:

In Subclause 11.48, "<abstract data type body>", Syntax Rule 26)a) defines the implicit "equals" function of an ADT when no explicit function is provided.

Unfortunately, this means that there is no way for an ADT that is a subtype to inherit an equals function from a supertype.

315 YOK-107/X3H2-93-197/MUN-030 contained the following problem:

The way domains are now defined, following SLC-040 and SLC-047, the association between any stored value and its domain (if any) cannot survive a <column reference>, the result of which does not have a domain. It is therefore not easy to see how it can ever be established whether two <column reference>s have the same <enumerated type>, and if it can't be establisehd, one cannot decide whether they are comparable.

318 During discussion of MUN-066/X3H2-93-139 and MUN-061/X3H2-93-070, somebody in the Munich ISO DBL RG meeting noted the following problem:

After the acceptance of MUN-061/X3H2-93-070 and postulating the future acceptance of something like MUN-066/X3H2-93-139, what would be the values of the DYNAMIC_FUNCTION field in the diagnostics area after the execution of such a facility?

319 This Possible Problem has been moved and is now Possible Problem **PSM-022**.

320 X3H2-93-096/MUN-069R and X3H2-93-140/YOK-101/MUN-067 conflict, raising the following problem:

MUN-069R/X3H2-93-096 completely rewrote Subclause 6.3, "<item reference>", deleting the former Subclause 6.6, "<column reference> and <row reference>", in the process.

However, paper MUN-067/X3H2-93-140, whose intent relaxed the restriction on having exactly one procedure with an OPEN statement, depended on a Syntax Rule in the former Subclause 6.3, "<item reference>" that does not appear in the rewritten version.

This problem has to be resolved.

321 During the Munich discussion of X3H2-93-305R/MUN-076R, Peter Pistor raised the following problem:

According to the current base document, the functions:

```
INTERSECT (X LINE, Y POINT) RETURNS...
```

and

```
INTERSECT (Y POINT, X LINE) RETURNS...
```

are two different functions. With a function invocation using keyword syntax, *e.g.*:

```
INTERSECT (X => XX, Y => YY)
```

it is not clear which of the two functions is to be invoked.

322 Paper MUN-082/X3H2-93-483 raised the following problem:

The absence of any General Rules describing the UNDER privilege is a problem.

323 During the Munich discussion of MUN-081/X3H2-93-483 and MUN-193, the following problem was noted by Nelson Mattos:

Column descriptors do not have the original name of an inherited column in the supertable.

324 MUN-081/X3H2-93-483 and MUN-193 contained the following problem:

Consider the literal interpretation of General Rule 7) of Subclause 11.15, "<add column definition>" ("If T is a supertable..."). Executing the <add column definition> on each of those subtables would cause General Rule 6) to be executed on those subtables, thereby adding an originally-defined column—which is clearly incorrect if we try later to drop the column.

Furthermore, for views, etc., we had a rule that said that schema changes effected using SML (e.g., added columns) didn't affect previously-defined objects (e.g., views). Are we happy that this isn't true for subtables/supertables?

325 MUN-159/X3H2-93-331R/X3H2-93-394R2 contained the following problem:

The use of collections in places where tables can be used in SQL needs to be clearly specified. For example, INSERT, UPDATE searched, and DLETE all require a <table reference>, which can currently only reference a collection variable by using a clumsy subquery.

326 During discussion of MUN-101/X3H2-93-404R, Hugh Darwen raised the following problem:

Effects of nulls in ordering and equality of ADTs. $A>B$ must be determined by evaluating $A<B$, then possibly $A=B$, says MUN-101/X3H2-93-404R.

This gives an ugly and sinister asymmetry, for it can happen that $A>B$ is false while $A=B$ is null and $A<B$ is null. Yes? In that case, surely we need user-defined "greater than" functions, too. (Reminder: Nulls always bite!) Problem goes away if null attributes are prohibited.

329 MUN-128/X3H2-93-514 raised the following problem. Note that this problem applies to SQL-92 as well.

Syntax Rules 14, 15, 16 and 24 of Subclause 5.4, "Names and identifiers", in SQL-92 state that the data type of the <simple value specification> that pertains to the subject(s) of the rule must be character string with an octet length of 128 octets or less. This length constraint raises several questions as follows:

a) Why is the restriction in terms of octets rather than characters?

b) What does the constraint imply for an embedded host variable? Does it mean, for example, that the data type of the variable cannot be CHAR(129) even though there might, at run time, be several trailing spaces? If so, does it also mean that the data type cannot even be VARCHAR(129)?

c) As all the subjects except <SQL-server name> are actually stripped of leading and trailing spaces at run-time before being validated as legal identifiers (including the literal cases as well, where allowed), why is this constraint needed in 5.4 for these cases?

It seems to me that all the cases with an existing run-time check should have this 128 octets constraint deleted from 5.4 and for <SQL-server name> (in SR24) it would be more sensible to have a suitable run-time constraint instead.

330 MUN-128/X3H2-93-514 raised the following problem. Note that this problem applies to SQL-92 as well.

GET DESCRIPTOR and GET DIAGNOSTICS retrieve information into host variables. However, as the values retrieved are not SQL-data values, the rules of Subclause 9.1, "Retrieval assignment", do not apply. But if these rules do not apply, where are the applicable rules? Similarly, SET DESCRIPTOR stores values into a descriptor area, but as the values that are stored are not SQL-data values, the rules of Subclause 9.2, "Store assignment", do not apply. But if these rules do not apply, where are the applicable rules?

331 MUN-128/X3H2-93-514 raised the following problem. Note that this problem applies to SQL-92 as well.

Table 17 in Subclause 17.1 in SQL-92 gives the data types of the fields in an item descriptor area. When setting or retrieving these fields, the corresponding host variable must have the same data type. Table 21 in Sublause 18.1 in SQL-92 serves a similar purpose for the diagnostics area in respect of retrieval; however, there is a significant difference in the way character string fields are described.

In Table 17, a character string field is typically defined as "character string with character set SQL_TEXT and length not less than 128 characters", whereas in Table 21 a character string field is typically defined as "character varying (L)" where L is an implementation-defined integer not less than 128.

I don't understand why these differences exist and, in particular, I don't see how the diagnostics area fields can be defined as character varying when not all host languages support host variables that equate to character varying.

332 MUN-128/X3H2-93-514 raised the following problem. Note that this problem applies to SQL-92 as well.

SQL Technical Corrigendum number 1 makes two changes to 17.1 as follows to cater for implementation-defined data types:

a) Adds SR2l):

21) TYPE indicates an implementation-defined data type and T satisfies the implementation-defined rules for matching that data type.

b) Adds SR3i):

i) TYPE indicates an implementation-defined data type.

Item a) above enables implementation-defined data types to pass the *match* criteria and item b) above enables them to pass the *valid* criteria. Before addressing my main concern, I think it is worth asking why item b) does not include some statement about the other fields having valid values according to the implementation-defined rules for that data type.

Items a) and b) are indeed needed but I believe that they overcome only two of the three hurdles that exist for implementation-defined data types. Without item a), you can't set the DATA field and without item b), you won't get very far when executing a <dynamic open statement>, <execute statement> or <dynamic fetch statement>. However, the third hurdle is the implicit CAST that takes place for all three of these dynamic statements. The table of valid source and target combinations in 6.10, <cast specification>, does not cater for implementation-defined data types, so the implicit CAST will always violate the SRs of 6.10 and SQLSTATE 07006 will always occur.

333 During X3H2 discussion of MUN-120/X3H2-93-486, David Beech raised the following problem:

Subclause 11.7, "<attribute definition>", Syntax Rule 2) ("If the <attribute definition> is contained in an <abstract data type definition>, then let *ADT* be the abstract data type identified in that <abstract data type definition>. The <attribute name> in the <attribute definition> shall be different from the <attribute name> of any attribute of *ADT*.") is ambiguous.

335 X3H2-93-486/MUN-120 noted the following problem:

Several places in the text dealing with holdable cursors state that a holdable cursr remains open after a commit; nothing is said about the last SQL-transaction in an SQL-session. Presumably, such cursors are closed then. Perhaps a rule should be added either to Subclause 4.41, "SQL-sessions", or to Subclause 15.3, "<disconnect statement>"?

336 X3H2-93-494 noted the following problem:

Assignment behavior may be unexpected when the target is an INSTANCE attribute. If Dick's spouse is assigned to be Jane's manager, then Dick's spouse will be equal to Jane's manger. However, if Jane's mangaer is assigned to be Dick's spouse, they will not be equal.

This discussion depends on the table definition:

```
CREATE TABLE EMP
       ( PERSON_DATA PERSON,
         MANAGER     PERSON,
         SPOUSE      PERSON INSTANCE,
         ... )
```

338 X3H2-93-488/MUN-122, part 4, noted the following problem:

Privileges for modules seem to be rather a mess. Clause 10.3, <routine invocation> currently (and with the change proposed above) requires EXECUTE privileges on the containing Module. EXECUTE privileges are grantable for a <module> but not created by clauses 13.1, <SQL-client module definition> [ANSI only] or 13.2, <module>.

GR 3 [Case: a) If a <module authorization ...>] of clause 13.2, <module>, which is a throw-back from SQL-92 defines an auth-id for privilege determination but this is clearly not sufficient now we have EXECUTE privileges on modules.

Clause 11.2, <SQL-server module definition> in MUN-006, GR 3 [A privilege descriptor ..] puzzles me a little. It does not seem to be fully compatible with the rule specified above (which was derived from the SQL-92 text). As the rule developed from the splitting of the document (i.e it was not present in any form in the pre-split SQL3 document) perhaps it warrents some discussion; possibly the rule is meant to generate an additional privilege descriptor; possibly it is redundant given the proposed change above. If no decision is made following discussion, then an entry should be included in the list of possible problems.

The proposed solution in the paper may be incompatible with SQL-92.

339 This Possible Problem has moved and is now Possible Problem **PSM-023**.

340 This Possible Problem has moved and is now Possible Problem **PSM-024**.

343 During the discussions of X3H2-93-490/MUN-124, Hugh Darwen noted the following problem:

Data types of empty collections

It is desirable to be able to compare, for example, SET() with a set of arbitrary element data typ. This would seem to require coersion rules, as we have for character strings for example, but none are given. In Clause 4.12 of MUN-003, we are told that "an empty set has data type SET()" in an attempt to achieve the desired effect, but htat is nonsense. There is no such data typ.

See point 6 on page 3 of MUN124, "Problems with Collection Types".

344 During the discussions of X3H2-93-490/MUN-124, Hugh Darwen noted the following problem:

Insufficient thought was given to collection type descriptors when YOK-035 introduced collection types into SQL3. in MUN-003:

a) GR10 of 6.1 <data type> says that a collection type descriptor is created. I'm not sure if this is right—we don't create descriptors for any other data types in the GRs of <data type>.

b) Collection type descriptors are apparently never used and never destroyed.

c) Collection type descriptors hve no provision for nested collection types.

d) A collection type descriptor includes "the name" of its element data type. A data type doesn't have a unique name, in general.

See point 5 on page 3 of MUN-124, "Problem with collection types".

346 X3H2-93-463/SOU-???, noted the following problem:

The definition of an abandoned view does not take into account loss of SELECT privilege against columns.

347 X3H2-94-027/SOU-041, noted the following problem:

Replacement for Syntax Rule 10) in Subclause 5.4, "Names and identifiers", assumes you know whether it's a type or a domain, but we don't know that at the time the statement is parsed.

348 X3H2-94-027/SOU-041, noted the following problem:

Name resolution for type templates is not taking into account the overloading of type template names.

349 Bill Kelley noted the following problem:

For collections types, referential integrity is not definable for elements of collections.

Example: Assume table EMPLOYEE has PRIMARY KEY EMP_ID of type INTEGER:

```
CREATE TABLE MANAGER UNDER EMPLOYEE
( MANAGES SET (INTEGER) );
```

Here "MANAGES" refers to a set of employees, but there is no way to say that they should reference employees. That is, one can't say something like:

```
(MANAGES SET (INTEGER REFERENCES EMPLOYEE) )
```

(Editor's note: In my opinion, Bill is simply trying to solve the problem using the wrong tools. SET(INTEGER) is meant to have sets of integers, not sets of employee IDs...which is a different thing altogether. The MOOSE stuff we've been developing solves this sort of problem.)

350 Phil Jones noted the following problem:

According to the Concepts, an empty set has the data type SET( ). Also, according to the Concepts, a set value of type SET(*DT*) retains its data type even when it becomes an empty set. There seems to be a contradiction. A similar contradiction exists in multiset type and list type.

351 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-008) noted the following problem:

The Restricted type definition in SAIF provides a number of types of restricted clauses that allow a user to specify a single value or type, a set of values or types, or a range of values. This approach is handled in some cases by SQL3 integrity constraints; however, some extensions

are required to encompass the full set of SAIF options. SAIF provides greater control through a richer syntax that permits specification of a check against a named type (^type), the use of OR and NOT to permit a check against a List, Set, or Enumerated type, and the use of a wildcard option to simplify specification of a check against a lengthy hierarchy or list, or when polymorphism may affect the syntax. It is possible to provide the same constraints in SQL, but the construction can be quite unwieldy.

352 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-008) noted the following problem:

SQL3 should support enforcement of non-instantiability, as is the case with SAIF's Abstract SuperTypes. With non-instantiable types, it should be possible to provide no attributes and no constraints, only attributes, or only constraints. A current workaround for this problem is to use a provate constructor function.

353 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-008) noted the following problem:

In the SQL context, it is often undesirable to pass constant values to a large number of instantiated objects (such as is described by the SAIF class attribute value type). For example, an object defined to describe a fire hydrant may contain a class attribute value for the color attribute (always red). It would be unnecessary and undesirable to pass this value each time the fire hydrant type is instantiated. Typically, SQL would define a single instance that contains the constant values for fire hydrants. One possible solution to this problem is to define a "generic" fire hydrant type that does not contain the color attribute is used for each instance. This implies the requirement of a Union construct that in practice would permit the generic and specific objects to reference shared attributes and methods. A Union construct may also be able to support heterogenous sets and lists.

354 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-008) noted the following problem:

There are situations where it is desirable to allow a list or set to contain instances of different types. For example, a set of urban structures may include telephone poles, fire hydrants, subway stations, and the like, each coming from a different type with different attributes and constraints. Such heterogeneity is permitted in SAIF, but currently not in SQL3.

355 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-008) noted the following problem:

Enumeration in SAIF is an ordered collection of named values that permit the user to refer to values by their common (everyday) names, without the need to map then onto the data type of a particular programming language or database. An enumeration describes all possible values that may be assigned to the attribute in question. This is consistent with the EnumeratedType in SQL3. As one of the three fundamental types in the SAIF Data Model, it is hoped that this type will be retained in SQL3.

356 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-008) noted the following problem:

SAIF defines an AbstractObject with an internal identifier as the root type for Tuples and Aggregates. SQL3 does not provide a root type.

357 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-008) noted the following problem:

Class attribute values as defined in SAIF describe a value for a class attribute that is not a default but a constant. Typically, a class attribute value is specified as a single value from a primitive type or a description from an enumeration. In the case of the definition of an abstract superclass, a single value may be defined, as well as a set of values or a ranget of primitives. In either of these cases, any subclass that may be instantiated directly must have for each class attribute a single value from this range or set, defined in its class definition or inherited form a superclass. SQL3 does not currently have an equivalent concept to class attribute value.

358 X3H2-94-037/MUN-189 noted the following problem:

GEO_Cell3DReference is defined using the enumerated predefined data type. SQL3 does not permit this use of predefined types. In particular, it is not clear how to do this for enumerated types. That is, do you have to define a domain for the enumerated type and then use the domain?

359 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-009) noted the following problem:

FUNCTION GEO_Neighbourhood in CREATE TYPE GEO_GeographicObject is requires a CHECK clause. SQL3 does not permit CHECK clauses on FUNCTIONs. They are only permitted on attributes.

360 X3H2-94-037/MUN-189 noted the following problem:

GEO_TemporalObject is a UNION based on the fact that the time attribute can have one of the following values: Time, Date, or TimeStamp. Functions like GEO_Equals will have to test this attribute to deteremine the type of GEO_TemporalObject but there is no way to do this in SQL3.

Steve Cannan recommends that we use the same technique as used in the SQL Information Schema. Use a flag and n different attributes where the UNION has n possibilities. The flag determines which attribute to use and $(n-1)$ of the attributes are always NULL.

361 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-010) noted the following problem:

OilSpill is defined in SQL/MM YOK-010 as:

```
CREATE TYPE OilSpill UNDER GEO_GeographicObject, GEO_TemporalObject
       ( spillIdentifier     CHAR(30),
         viscosity           INTEGER,
         chemistry           REAL,
         temperature         REAL,
         velocity            REAL,

         CONSTRAINT OilSpill_C1 CHECK (Is_GEO_Polygon(geometry)),
         CONSTRAINT Oilspill_C2 CHECK (metadata IS NULL) )
```

While the syntax of this definition is correct, the semantic checks would fail since the objects of the OilSpill type belong to two separate type hierarchies. SQL3 does not currently prohibit this syntax but on the other hand neither does it support the intent.

362 X3H2-94-037/MUN-189 (referencing SQL/MM YOK-010) noted the following problem:

OilSpill is defined in SQL/MM YOK-010 as:

```
CREATE TYPE OilSpill UNDER GEO_GeographicObject, GEO_TemporalObject
       ( spillIdentifier     CHAR(30),
         viscosity           INTEGER,
         chemistry           REAL,
         temperature         REAL,
         velocity            REAL,

         CONSTRAINT OilSpill_C1 CHECK (Is_GEO_Polygon(geometry)),
         CONSTRAINT Oilspill_C2 CHECK (metadata IS NULL) )
```

geometry is a GEO_GeographicObject attribute and not an attribute of OilSpill. Using this attribute in the CONSTRAINT breaks the principal of substitutability, since functions that are defined on GEO_GeographicObject that assign values to geometry may not work for OilSpill. SQL3 allows this, but Nelson Mattos is not sure that this is a good feature since it requires run-time checking of the data types.

Note that this problem occurs in other definitions (e.g., GEO_Boundary).

363 X3H2-94-037/MUN-189 noted the following problem:

The GEO_GeometricAggregate definition uses REF(MULTISET()) which is not supported by SQL3.

364 Discussions on X3H2-94-???/MUN-156R1 noted the following problem:

There is a problem for precompilers when the issue of overlapping and non-disjoint scopes for host variables, etc. comes into play. In addition, there are problems caused by things like C macros and the C #ifdef conditional facilities.

365 X3H2-94-???/MUN-132 noted the following problem:

What is a "database management system" in SQL terms?

366 X3H2-94-???/MUN-132 noted the following problem:

How are the boundaries of SQL database management products mapped into the SQL concepts? That is, how can a product claiming conformance interact with another product that is also conforming? There seems to be a confusion in SQL between environment and implementation.

367 X3H2-94-???/MUN-132 noted the following problem:

Whether or not an SQL-environment contains a complete set of the catalogs that make up a catalog cluster. If all the catalogs in a cluster are cross-related for definitional or constraint reasons, then I believe that we can conclude that all catalogs in the cluster must be contained in the SQL-environment; otherwise, insufficient information is available to check every action that may be requested for legitimacy.

368 X3H2-94-???/MUN-132 noted the following problem:

Whether or not an SQL-environment can contain more than one catalog cluster.

369 X3H2-94-???/MUN-132 noted the following problem:

Whether or not an SQL-server can access more than one catalog cluster.

370 X3H2-94-???/MUN-132 noted the following problem:

Since SQL-data from two catalogs not defined in the same cluster cannot be accessed in the same SQL-session, why cannot the definintion of SQL-data simply be:

> SQL-data is all the data described by all the schemata of all the catalogs in one cluster.

371 X3H2-94-???/MUN-132 noted the following problem:

Mapping of Direct SQL onto the module language.

372 X3H2-94-???/MUN-132 noted the following problem:

What terms should SQL use to describe teh data defined by an SQL-schema and an SQL-Catalog?

373 X3H2-94-???/MUN-132 noted the following problem:

If a conforming SQL-Client is to be able to interact with any conforming SQL-server from another supplier than the interface between SQL-client and SQL-server needs to be defined in at least one standard or a defined combination of standards.

374 X3H2-94-???/MUN-132 noted the following problem:

I believe that the <test completion statement> is incorrectly categorized in the Working Draft. I believe it to be an <SQL control statement> and not an <SQL transaction statement> as it merely loads the diagnostic area when an axynchronous SQL statement has completed and does not affect the transaction in any way.

375 X3H2-94-???/MUN-132 noted the following problem:

Now that we have control statements and multi-statement procedures, I believe that the model of the diagnostics area is incorrect. Subclause 4.42, "Client-server operation", requires that an <SQL diagnostics statement> be executed at the client end. Given that all three diagnostics statements can, and at least one must, be part of a compound or multi-statement procedure that is executed at the server side, I believe that the diagnostics area must exist wherever the execution is taking place and be passed back to the client when the routine is complete. The diagnostics area does not need to be passed forward as no routine that executes at the server fails to initialize the diagnostics area. The only statement that leaves it alone is <get diagnostics statement>, which may be executed at either end.

376 X3H2-94-???/MUN-132 noted the following problem:

The RMDM does not seem to recognize that an SQL processor may, within the same transaction, manipulate for the data and the metadata. Indeed, a number of the constructs, e.g., <temporary table declaration>, require both to be possible in order to have any use.

377 X3H2-94-???/MUN-132 noted the following problem:

The CLI Working Draft does not have a Conformance Clause.

378 X3H2-94-???/MUN-132 noted the following problem:

Usage of RPT as an SQL-client/SQL-server communications ethod and the resolution of the transfer of the SQL diagnostics area in such a schem.

379 X3H2-94-???/MUN-132 noted the following problem:

The different SQL-diagnostics model of the SQL CLI Working Draft.

(Editor's note: This has been corrected in the most recent revision of SQL/CLI.)

370 X3H2-94-???/MUN-132 noted the following problem:

Can a schema span SQL-servers?

380 X3H2-94-???/MUN-132 noted the following problem:

Can a catalog span SQL-servers?

381 X3H2-94-???/MUN-132 noted the following problem:

Can a "schema" module contain connection statements? Diagnostic statements?

382 X3H2-94-???/MUN-132 noted the following problem:

Where to non-schema modules reside? At the SQL-client?

383 X3H2-94-???/MUN-132 noted the following problem:

If a module resides at an SQL-server, how and when is it shipped to the SQL-client? Is it shipped to the SQL-client? If not, how are connection and diagnostic statements handled?

384 Discussion of X3H2-93-110/X3H2-93-239/MUN-063 led to Hugh Darwen providing the following problem:

<ordering clause> is defined in Subclause 11.48, "<abstract data type body>", of SOU-003, starting on page 467. According to SR 19), no more than one <member> of an <abstract data type body>'s <member list> can be an ordering clause, so the first problem is that the syntax doesn't permit both an EQUALS function and a LESS THAN function to be specified. The second problem is closely related to the first, and is alluded to in the Editor's Note following SR 18): The combination of SR 17) and SR 18) is a contradiction. If no <ordering clause> is present, then under SR 17), you get an implicit one that specifies and EQUALS function, while under SR 18), you get an implicit one that specifies a LESS THAN function. Thus, two <ordering clause>s are yielded, which is prohibited, as already noted, under SR 19).

While the first two problems can no doubt be easily corrected by splitting <ordering clause> in to <ordering clause> (for LESS THAN) and <identity clause> (for EQUALS), the third problem is more difficult. Suppose I create type MANAGER UNDER EMPLOYEE, and the definition of EMPLOYEE specifies carefully constructed user-defined functions to determine equality and ordering of EMPLOYEE instances. Under the SRs 17), 18) and 19) of Subclause 11.48, "<abstract data type body>", if I want MANAGER instances to be compared using EMPLOYEE's EQUALS and LESS THAN functions, I must write explicit EQUALS and LESS THAN clauses specifying exactly the same <specific routine designator>s as in the definition of EMPLOYEE. Furthermore, if I later modify the definition of EMPLOYEE to use different <specific routine designator>s, I must remember to modify the definition of MANAGER too.

This is totally contrary to the inheritance rules that apply to user-defined functions in general.

385 The Editor has noted the following problem with the BNF of SQL3:

The BNF nonterminal "<operator name>" is defined in a production in Subclause 5.4, "Names and identifiers", but is used *only* in the General Rules of that same Subclause. What is the purpose of this nonterminal?

386 The Editor has noted the following problem with the BNF of SQL3:

The BNF nonterminal "<function invocation>" is defined in a production in Subclause 6.2, "<value specification> and <target specification>", but is never used. What is the purpose of this nonterminal?

387 The Editor has noted the following problem with the BNF of SQL3:

The BNF nonterminal "<alter schema statement>" and a subsidiary BNF nonterminal "<add operators definition>" were created by X3H2-93-383/MUN-097, but are not actually used anywhere. The proposal apparently failed to "put" the new SQL-statement into the BNF production for <SQL procedure statement> or into the Concepts where statements are categorized.

389 The Editor has noted the following problem with the BNF of SQL3:

The BNF nonterminal "<qualifier>" is used in several places in the document, notable in the Format of Subclause 6.4, "<row reference>", as part of the definition of the BNF nonterminal "<row reference>", and in the Format of Subclause 7.13, "<query specification>", in the definition of "<select sublist>". However, "<qualifier>" is never defined anywhere in the document.

390 The Editor has noted the following problem with the BNF of SQL3:

X3H2-93-444/MUN-167 used the BNF nonterminal "<data type name>" in the Format of Subclause 11.52, "<drop data type statement>", as well as in the Syntax Rules of that Subclause. However, that nonterminal is not defined anywhere in the document. Should this really be "<abstract data type name>"?

391 In the course of discussing X3H2-94-201/SOU-184, Nelson Mattos noted the following Possible Problem:

> SQL3 does not specify the value returned by a function body that does not execute a return statement.

392 In the course of discussing X3H2-94-126R1/SOU-060, Hugh Darwen noted the following Possible Problem:

> Syntax Rule 6) of Subclause 7.14, "<query expression>" (Sytnax Rule 4) of Subclause 7.9, "<query expression>", in SQL-92), is not sufficiently precise where it attempts to exclude common columns of a joined table from the expansion of Q.*. A consequence is that, for example, SELECT X.* FROM (T NATURAL JOIN U) AS X would exclude the common columns of the join, and thus be different from SELECT * FROM (T NATURAL JOIN U) AS X.

393 In the course of discussing X3H2-94-027/SOU-041, Nelson Mattos noted the following Possible Problem:

> The current document cannot resolve user-defined type and domain names. Format rules like
> `{ <data type> | <domain name> }`
> should be modified in such a way that the given name is first resolved. Once resolved, a domain or user=defined type can be identified by the name.

394 In the course of discussing X3H2-94-031R1/SOU-045, Hugh Darwen noted the following Possible Problem:

> The change proposed by X3H2-94-031R1/SOU-045 does not permit an <interval qualifier> of SECOND TO SECOND, although that appears to be in the spirit of things. It was apparently accidental that it was prohibited, and this should be corrected.
> Worse, if SECOND TO SECOND is permitted naïvely, then SECOND(4) TO SECOND(6) and variations thereof would also be permitted, but that doesn't appear to be very meaningful and should be prohibited.

395 In the course of discussing X3H2-94-088/SOU-048, Steve Cannan noted the following Possible Problem:

> Syntax Rule 6 ("If any <column name>...specify NONE.") of Subclause 11.11, "<unique constraint definition>", is redundant, as <equals clause> of an ADT does not now have NONE as an option.

396 Paper X3H2-94-293/SOU-146 noted the following Possible Problem:

> The definition of the implicit accessor and mutator functions do not satisfy the Format for a <routine declaration> since they have no <routine body>. Changes to the Format, Syntax Rules, and General Rules may also be necessary to allow the definition of such routines, because it is not even in principle possible to write such a body without infinite regress.

Paper X3H2-94-259/SOU-111 further commented:

> In Subclause 11.48, "<abstract data type body>", Syntax Rule 26), ("Let *ADTN* be the <abstract data type name>..."), the proposed text of the implicit <routine declaration> contains the string "..."; this is not legal syntax, nor does the rule specify how and with what it should be substituted.

398 Paper X3H2-94-239/SOU-051 noted the following Possible Problem:

In examining the Access Rules of Subclause 6.3, "<item reference>", to see if anything should be changed in the light of <query name>s, we noticed that privileges are required on all columns that can be referenced by an <item reference> that is a column reference (an undefined term, now). We wonder if this is correct, in consideration of the fact that there is no GRANT statement that could give SELECT privilege on, for example, the column T.C in "SELECT T.C FROM (SELECT 'Yes' FROM T) AS T".

399 Discussion of paper X3H2-94-240/SOU-052 noted the following Possible Problem:

The use of the term "a value of an abstract data type" in the document (for example in Subclause 4.11, "Abstract data types") is in conflict with the definition of value (of an ADT instance) which is defined in Subclause 3.1, "Definitions". The term "a value of an abstract data type" would not deal with "values of ADTs" that are intended to mean "an instance of an object ADT".

400 Paper X3H2-94-108/SOU-079 noted the following Possible Problem, augmented by a note from Nelson Mattos:

This Working Paper therefore outlines a proposal to delete the single maximal supertype rule, to get feedback concerning possible problems with the proposal. As a result of dsciussion of this paper in ANSI, Amelia Carlson noted the following: "Consider the use of 'subtype family' in Subclause 9.3, "Set operation result data types and nullabilities", Syntax Rule 3)i):

i) If any data type in *DTS* is an abstract data type, then each data type in *DTS* shall be an abstract data type that is in the same subtype family. Let *ADT* be the data type that is the most specific unique type that is a supertype of each data type in *DTS*. The result data type is *ADT*.

If you remove the restriction that there is a single maximal supertype, then you can have overlapping subtype families for two types; by overlapping, I mean there will be times when neither sutype family completely contains the other; when one subtype family contains the other; and when they are the same. So, rules such as the above, which check that two ADTs are in *the same* subtype family, will need to be modified to check if they are in the common part of two overlapping subtype families, and deal with cases where there is more than one most specific unique type that is a supertype of the data types involved. There can be more than one because they can occur at the same level; a single maximal unique aupertype prevented the overlapping and multiple most-specific unique type cases from occurring.

Function invocation and copmparison predicates may also suffer from this problem, and 'subtype family' is also used in function resolution."

Nelson Mattos added: The definition of subtype family needs to be modified to make clear that subtype families are defined in terms of a maximal supertype. Consequently, a "non-maximal supertype" may belong to more than one subtype family. However, a maximal supertype defines a single subtype family that contains all its subtypes.

401 Discussion of paper X3H2-94-143/SOU-086 noted the following Possible Problem:

Does Subclause 14.1, "<start transaction statement>", Syntax Rule 1) permit things like "START TRANSACTION DIAGNOSTICS SIZE 10 DIAGNOSTICS SIZE 20"? That is, are "DIAGNOSTICS SIZE 10" and "DIAGNOSTICS SIZE 20" the *same* <transaction mode>, or are they *different* <transaction mode>s? Similarly, in Syntax Rule 1) of Subclause 16.1, "<set session characteristics statement>", is "SET SESSION CHARACTERISTICS AS TRANSACTION READ ONLY TRANSACTION READ WRITE" prohibited or permitted?

404 Paper X3H2-94-285143/SOU-155 noted the following Possible Problem:

[Amelia] then constructed an example along the following lines.

1) Two type templates are defined in the same schema along the lines of the following:

```
CREATE TYPE TEMPLATE pick_me (:p1 TYPE, :p2 INTEGER) (...);
CREATE TYPE TEMPLATE pick_me (:p1 INTEGER, :p2 TYPE) (...);
```

2) Then in schemas/modules/compound statements with the same name, [Amelia] create[s] a variable and an ADT:

```
DECLARE pink_elephant VARIABLE FOR INTEGER;
CREATE TYPE pink_elephant (...);
```

3) Finally, from somewhere that sees those templates and that type, [Amelia does] the following:

```
DECLARE which_one VARIABLE FOR pick_me (pink_elephant, pink_elephant);
```

This example led me to the following burning question:

Which one of the type templates should be picked.

405 Discussions in the joint DBL/RDA meeting held in Southampton prompted the following Possible Problem (noted by Bob Sunday among others):

RDA encountered a technical problem for which there is no known resolution: character sets must be identified by an OSI object identifier, but there are no such object identifiers defined, specifically for user-defined character sets. There must be a correspondence between SQL's names for character sets and OSI object identifiers.

406 Paper X3H2-94-244/SOU-097 noted the following Possible Problem:

In Subclause 6.3, "<item reference>", Syntax Rule 5), Subrule a) starts: "If *V* is a <table name> or a <correlation name> ...", and Subrule b) starts "If *V* is a <routine name> ...". However, *V* is clearly not any sort of name. Various rules, in particular Syntax Rule 3) ["If IR contains ..."] define *V* to be a table or a parameter list.

407/BOLD Paper X3H2-94-244/SOU-097 noted the following Possible Problem:

Subclause 6.17, "<numeric value expression>", Syntax Rule 2) ["If a <factor> F ..."] contains the phrase "and the data type of the <numeric primary> NP simply contained in F is an abstract data type ...". However, according to Syntax Rule 6) ["The data type of a <numeric primary>..."], the data type of a <numeric primary> cannot be an abstract data type. Further more, all the General Rules in this clause are oriented towards describing normal mathematical operations and clearly may not apply if "+" is some arbitrary user-defined function.

408 Paper X3H2-94-244/SOU-097 noted the following Possible Problem:

In Subclause 6.18, "<string value expression>", Syntax Rule 1) ["If one or both operands ..."] contains the phrase "one or both operands of a <concatenation> is an abstract data type ...". However, according to the format, one of the operands is a <character factor>, which is in turn a <character primary>, the data type of which is character string (see Syntax Rule 2) ["The data type ..."]). The other operand of a <concatenation> is a <character value expression>, the data type of which, according to Syntax Rule)4 is also character (either explicitly or because it inherits the data type of its <character factor>, which is its self character. Furthermore, all the General Rules are expressed as the result of string operations which clearly may not apply to user-defined functions operating on abstract data types.

409 Paper X3H2-94-244/SOU-097 noted the following Possible Problem:

Default values for attributes?

In MUN-003, the <default clause> was removed from the production for <stored attribute> (Subclause 11.7, "<attribute definition>").

Firstly it should be noted that in Subclause 4.19, "Columns, fields, and attributes", still states that an attribute descriptor includes:

> "for a stored attribute, the value of <default option>, if any of A".

Probably that sentence should be removed.

Secondly however, <stored attribute> can still be based on a <domain>, which can have <default clause>. Perhaps its clear by omission (in that nothing is said about what happens with the domain's default value when the domain is applied to an attribute) but in that case it would be applicable to say that with an informative Note. Otherwise, if we believe that a rule is missing, then a PP should be raised.

410 Paper X3H2-94-244/SOU-097 noted the following Possible Problem:

<column constraint definition>s in <stored attribute>s: The production for <stored attribute>, Subclause 11.7, "<attribute definition>", contains the production <column constraint definition>. This looks a nonsense as <column constraint definition>s are transformed into <table constraint>s - see Syntax Rule 8) of Subclause 11.6, "<column definition>", which is hardly applicable in the case of attributes.

In YOK-114, which separated columns and attributes, constraints for attributes were indicated by the production <attribute constraint definition>, but this production was not further defined. Given the debate about the validity of constraints within ADTs, it seems the lesser of the evils to replace "<column constraint definition>" by "<attribute constraint definition>" in the production for "<stored attribute>, (Subclause 11.7, "<attribute definition>") and to add an Editor's note and PP saying that "<attribute constraint definition>" is an undefined production.

411 Paper X3H2-94-244/SOU-097 noted the following Possible Problem:

Collection type <literal>s:

Subclause 11.9, "<default clause>", Syntax Rule 4)a)ix) states:

> "If the subject data type is a collection type, then the data type of the <literal> shall be that collection type".

However, Subclause 5.3, "<literal>", specifies no literals of any of the collection types.

412 Paper X3H2-94-092/SOU-072 noted the following Possible Problem:

Many of the Rules in Subclause 6.7, "<table reference>", (e.g., Syntax Rule 11, "Let *T*...", all the Access Rules, General Rule 1, "The <correlation name>...") fail to cover all three possibilities now in the Format.

413 Paper X3H2-94-092/SOU-072 noted the following Possible Problem:

By the Rules of updatability in Subclause 7.13, "<query specification>", *V2* is not uniquely defined, since an updatable view may have more than one simply underlying table.

414 Paper X3H2-94-092/SOU-072 noted the following Possible Problem:

In Subclause 11.45, "<trigger definition>", Access Rule 4) is an Access Rule on creation of the trigger, intended to check that the creator has the privileges to execute the <triggered SQL statement>. The generalization of <triggered SQL statement> requires the rule to check all DML statements that might be executed, including those in invoked routines.

415 Paper X3H2-94-092/SOU-072 noted the following Possible Problem:

In Subclause 13.6, "<delete statement: positioned>", the General Rules has marking for deletion, but nothing actually gets deleted there (unlike Subclause 13.7, "<delete statement: searched>"). On the other hand, the General Rules of Subclause 7.13, "<query specification>", and Subclause 7.14, "<query expression>", used by the DELETE statements, actually delete without previously marking for deltion. Why is this?

416 Paper X3H2-94-092/SOU-072 noted the following Possible Problem:

In Subclause 13.6, "<delete statement: positioned>", and Subclause 13.7, "<delete statement: searched>", should there not be General Rules that invoke default destructors for column values that have ADT type?

Language Opportunities

004 Subclause 6.3, "<item reference>", Syntax Rule 3), says that a <column specification> must be within the scope of one or more <table name>s. It seems that, despite Syntax Rule 5) of Subclause 13.1, "<declare cursor>", the <column specification>s in an <order by clause> do not obey this scope rule. (Note, by the way, that SR3)a) uses a different form of wording that SR3)b)–for example, "shall appear within" rather than "shall be contained within". Is this supposed to be significant?) X3H2-88-294, which was rejected, addressed this issue. Phil Shaw has agreed to address this issue

013 The "algorithm" for determining precisely *what* is an Access Rule is ill-specified; that makes it hard to know whether it's been done properly. Therefore, 1) Conventions and/or Concepts should be augmented to describe the criteria better, and 2) there are almost certainly rules that should have been moved to Access Rules that haven't been. Phil Brown will address this. This has been discussed by SLC-49 and significantly reduced in significance by X3H2-90-035/SLC-25.

014 It was noted in conjunction with CAN-106 discussions that if one inserts a row in a view V1 but do not have INSERT privilege on the underlying view V2 that has a WITH CHECK OPTION constraint, then a *constraint violation* exception is raised; however, one can then not discover anything about that constraint!

023 There is a possible problem related to cascaded integrity actions on cursor fetches. Consider the table T of X3H2-89-197:

```
CREATE TABLE T
       ( A SMALLINT PRIMARY KEY,
         B SMALLINT REFERENCES T ON UPDATE CASCADE )
```

```
 A | B
---|--
 1 | 1
 2 | 1
```

and the following statements:

```
DECLARE K CURSOR FOR
       SELECT * FROM T
         WHERE A >= B
OPEN K
loop until all rows fetched
FETCH K
UPDATE T SET A = A+2 WHERE CURRENT OF K
repeat loop
```

If row 1 is fetched first, then row 2 will never be fetched and the table will be:

```
 A | B
---|--
 3 | 3
 2 | 3
```

after the last update or else a row is fetched that does not satisfy the WHERE clause. But if row 2 is fetched first, both rows will be fetched and the table will be:

```
 A | B
---|--
 3 | 3
 4 | 3
```

This description is affected by cursor sensitivity. There is an analogous situation for ON DELETE CASCADE.

It should be noted that the CASCADE ON/OFF capability affects this, but that is in SQL3. Phil Shaw has agreed to address this item.

See also Possible Problem **183**.

040 The concepts section needs to explain that CAST AS is the mechanism to translate datetime and interval data types to and from host data parameters. In general, CAST AS probably needs to be discussed in the concepts section. Jon Kerridge has agreed to address this issue.

049 It was suggested in ISO DBL RG CAN-93 that privilege descriptors should contain a special value, such as "_PUBLIC", when describing privileges granted to PUBLIC. This would supersede the current restriction that requires that no authorization identifier may have the value "PUBLIC".

055 It has been noted that schema manipulation requires no privileges, but depends directly on ownership of the schema.

058 It has been suggested that <derived table> is now a redundant (superfluous) production and can be subsumed in the production for <table reference>. Steve Cannan believes that the references to <derived table> can be accommodated by references to <table subquery>.

061 There are too many reserved words. We should identify those keywords that must be reserved and those that need not be. That is, many keywords may be used only in limited contexts. In all other contexts, an identifier that is the same as one of those keywords could not be confused.

This suggestion means examining every BNF production, and where a keyword or identifier *may* be specified, the keyword must be a reserved word. Those keywords that are not reserved may be used as identifiers. The result would be that the list of <key word>s would be accompanied by a list of <reserved word>s and SR 9) of Subclause 5.2, "<token> and <separator>" would be changed to refer to <reserved word>s. **This last part has been done!**

062 The C language embeddings allows the retrieval of data containing the C NULL character (binary zero) into character arrays. There is no mechanism to insert from a character array data that legitimately contains the C NULL character.

067 The Schema Information Tables do not contain enough information regarding the relationship between subtables and gentables. See also Possible Problem **266**.

068 Paper FIR-95/X3H2-89-358 suggests "The unfortunate (unsmooth) syntax of <select statement: single row> requires that it be possible to decompose it into components that can be rearranged to form a <query specification>. It would be more general and consistent if it were defined in terms of a <query expression>.

070 Paper X3H2-89-376rev2/FIR-66 defines several new objects: character repertoires and translations. However, the CASCADE option has not been defined for the DROP statements for any of these objects. Without this, there is less security when the objects are dropped.

084 The new definition of substring could be improved by adding a standard warning - "warning - zero or substring starting position" (Ref: FIR-72) (Source: Jon Kerridge)

090 It might be desirable to delete the use of *identical* throughout the document and replace it with some well-defined term from Clause 4, "Concepts". One might possibly use *representationally different* and *representationally the same*. (Source: Jeff Mischkinsky)

092 This Language Opportunity has moved and is now Language Opportunity **PSM-025**.

116 Paper SLC-79 contained a paragraph that suggests: Clause 4, "Concepts", should contain a better description of the working of Dynamic SQL than it does at present. Subclause 4.8, "Embedded syntax", currently says nothing about statement identifiers or the way that they are used to identify the resources allocated with a statement is prepared. Such a description should be added.

117 Paper SLC-79, point 3, says: There is considerable overlap between Table 4, "Fields in datetime items", and Table 11, "Valid values for fields in datetime items", and also between Table 7, "Valid values for fields in INTERVAL items", and Table 12, "Valid values for fields in INTERVAL items". These pairs of tables should be merged. The merged tables should probably be positioned near Subclause 6.1, "<data type>", and consideration should also be given to relocating other related tables from Clause 4, "Concepts", to this general area. Detailed information is needed in Subclause 6.1, "<data type>", and redundant specification should be avoided.

129 We use the terms "destroyed", "deallocated", "deleted", "released", and perhaps others in various places. Are these terms used consistently and could the number of such terms be reduced?

130 Paper SLC-31/X3H2-90-031, item 2, gives rise to an Opportunity to make use of the terms *dependent on, depends on, is the determinant object for, include, be included in*.

131 The term "zero-valued bit" may not always be appropriate. A zero-valued bit is a bit (i.e., container) with a zero value and not a value itself.

132 The terms "left", "right", "least significant", "most significant", "begin", and "end" for character and bit strings may not be well defined. The use of ordinal position may be an improvement. The terminology should be made consistent and, if possible, not make use of cultural aspects of strings (e.g., Arabic vs Latin).

133 The document used to assign a label (e.g., S) to some object (e.g., the result of a <subquery>) and then used the label in SQL syntax (e.g., EXISTS S) in some places. This is syntactically incorrect. There may be other occurrences of the same problem.

134 The functions LOWER and UPPER might be better defined in terms of translations and collations so that they properly account for all character sets instead of only <simple Latin character>s.

152 Paper SEL-74, point 9, says:

> In the current specification, there is no specification of full SQL data type support in <module>. In <module>, there are two types of language dependence supported: one is the calling convention, and the other is data type conversion. [Shibano-san thinks] it is better to add "LANGUAGE SYSTEM, which supports full SQL data type and its calling convention would be implementation-dependent. This feature might be a good candidate for an RDA rrequirement of "language indenpendent" <module>s, because the RDA protocol machine and the SQL implementation both reside in a certain system that is the same system.

183 Pedro Celis submitted the following Possible Problem in his response to TC LB X3H2-90-266:

> Integrity. Referential actions always have statement level semantics. That is, the set of matching rows is always determined at the beginning of a SQL statement. The outcome of referential actions depends on how the application is coded: multiple cursor updates; one set update; or multiple single-row update statements. The programmer may choose one of the above due to unrelated reasons (e.g., printing old and new values, concurrency, ...). This choice should not affect the outcome.
>
> Just like the checking of referential constraints can be delayed to an arbitrary point in time (before transaction commit), it should be possible to delay the triggering of referential action until an appropriate time (like cursor close time). Alternatively, it should be possible to specify that referential actions are triggered on a row instead of statement basis.

See also **023**.

190 Jim Melton said, in his response to TC LB X3H2-90-267:

> We believe that many implementations will have schema objects other than those specified in SQL2 (e.g., indexes, stored <module>s, etc.) that may depend on schema objects defined in SQL2. The DROP semantics for such implementations will depend on those implementation-defined objects as well as those specified in SQL2, yet the SQL2 DROP rules do not appear to make allowances for additional restrictions on DROP statements. The wording in SQL2 must be enhanced to allow for such additional restrictions.
>
> Paper X3H2-90-373 addressed this, but failed. X3H2 suggested that a broader proposal that addresses the general concept of implementation-defined objects that might restrict CASCADE operations would be acceptable.

201 Jim Melton, in his response to TC LB X3H2-90-267, said:

> It is not clear that SQL2 currently partitions all session-specific objects correctly, especially when multiple sessions are invoked using the Association Management statements. For example, temporary tables are cited as being session-specific, but the Diagnostics Area is not discussed.

210 Ed Dee, in LON-93/X3H2-90-445) pointed out that <recursive union> still uses the term "description" instead of the more accurate "descriptor" and that this should be corrected.

212 The ISO SQL2 Editing Meeting in London noted that with the advent of a default character set for domains and columns in a schema, there is an opportunity to change that default character set for the schema. This might, for example, involve an ALTER SCHEMA CHANGE CHARACTER SET statement.

217 Steve Cannan has noted:

> It might be necessary to redefine the actions of triggers so that certain actions survive an *unsuccessful* execution of an SQL statement. For example, a BEFORE DELETE trigger might be used to record *attempts* to alter a table for security reasons. It would therefore be necessary that the triggered action survive an error in the original statement.

227 Steve Cannan has noted in LON-077/X3H2-90-429:

> In Subclause 6.1, "<data type>": Can we please have the rule that defined the BASIC character set back as this set (the <SQL language character>s) is referenced by name in at least 9 places. Also, it would seem a good idea to have the minimal set defined in the Information Schema.

241 The following Opportunity exists:

> When counting the number of rows "affected" by an <SQL statement>, one might consider counting the rows that are affected by triggered statements, too (e.g., triggers and referential constraints).

242 The following Opportunity exists:

> For language consistency, a correlation name should be permitted for the modified table in positioned and searched update and delete statements.

244 Jonathan Bauer notes the following opportunity in X3H2-91-012:

> The owner of a view can hold the UPDATE and/or INSERT table-level privileges, yet such privileges are meaningless. Since no columns can be added to a view, the fact that he/she holds this privilge is never acted upon.

247 Paper SLC-79, point 11, says: The styles of the two Leveling Rules in Subclause 8.1, "<predicate>", are different. [The author of SLC-79 prefers] that used in the first rule, and suggest[s] that it should be used throughout the document. We are stating a restriction on what constitutes the levels of SQL language, not direct restrictions on conforming source. The second style introduces, by implication, the unnecessary concept of non-conforming SQL language.

259 In CBR-017/X3H2-92-192, the following language opportunity was identified:

> X3H2-92-083/OTT-011, provided the SET keyword for the <assignment statement> and specified "compatibility with the UPDATE statement" as one motivation. Given that SET was chosen for compatibility with the UPDATE statement, shouldn't the statment allow multiple assignments, *e.g.*,
>
> SET a = b, c = d, ..., *etc.*
>
> as in the UPDATE statement?

260 During discussions of CBR-060/X3H2-92-236, the following language opportunity was identified:

> It might be desirable to find a way for ATOMIC <compound statement>s to also cause the values of parameters, session state (e.g., SET SCHEMA), and other "in-memory" items to come under atomicity control.

268 During consideration of YOK-023/X3H2-92-252, following language opportunity was identified:

> The set of <identifier>s available as <regular character set identifier>s in the <similar predicate> (see Subclause 8.6, "<similar predicate>") could profitably be enhanced to support additional character attributes (*e.g.*, *ideographs*, *syllables*, *etc.*, as a result of internationalization work subh as that going on in SC22/WG20.

275 During consideration of X3H2-93-066/YOK-035, the following language opportunity was identified:

> There is a language opportunity to make DESTRUCTOR functions and SET functions be arbitrary routines (i.e. they could also be procedures), since the value they return is not needed.

281 During consideration of X3H2-93-285/YOK-168, the following language opportunity was identified:

> The Concepts could be enhanced to clarify the fact that all SQL-transaction characteristics are reset to default values at the end of each SQL-transaction in an SQL-session, and that execution of a <set transaction statement> causes all attributes not specified to become the default (relevant when multiple <set transaction statement>s are executed before starting a transaction.

291 Paper X3H2-93-285/YOK-168 proposed the following language opportunity:

> In Subclause 11.16, "<alter column definition>", a Language Opportunity would be to permit the addition to or dropping from a column of a <null clause>.

309 The following Language Opportunity has been noted by Phil Shaw:

> Local declarations of dynamic cursor names would seem like a straightforward extension to X3H2-93-056/YOK-034rev.

311 The following Language Opportunity has been noted by the UK resulting from Email ballot #6/X3H2-93-098rev/YOK-084:

> The addition of "specific routine designators" to the Information and Definition Schemas is desirable.

317 The following Language Opportunity has been noted by X3H2-93-445/MUN-160:

> The representation of SQL-paths in the Information Schema needs to be specified.

327 The following Language Opportunity has been noted by X3H2-93-370R1/MUN-170:

> Object-oriented applications that model the behavior of real-=world entities need the ability to add an existing object to a type or to remove it from a type without destroying the object. Existing persons become employees and later stop being employees while continuaing to exist as persons. This can be achieved with a modest extension of current facilities.

The paper went on to add that a simple extension would be allow a constructor such as STUDENT() to accept an optional parameter whose value is an existing object that is to be made an instance of STUDENT (but only if it is in the type hierarchy with STUDENTs).

328 The following Language Opportunity has been noted by X3H2-93-372R1/MUN-171:

> The restriction that an object cannot have more than one most specific type creates several problems.
>
> One problem arises with the results of set operations. Consider the type graph:

```
          Person
          /    \
         /      \
      Student   Teacher
       |     \ /     |
       |      X      |
       |     / \     |
       Intern   Trainee
```

> in which Intern and Trainee are each subtypes of both Student and Teacher.
>
> Consider SET(Intern) UNION SET(Trainee). By the current rules, the only way to satisfy the requirement of a signle most specific type is to make the result type be SET(Person). However, all the instances of the resulting union will be students and this union should be acceptable wherever SET(Student) is required. The same holds for teachers and SET(Teacher).
>
> Under the current rules, any situation requiring SET(Student) or SET(Teacher) cannot accept SET(Intern) UNION SET(Trainee).
>
> Another problem concerns proliferation of subtypes. While tyis problem has probably been discussed before, it bears recording as a problem. There are potentially a great many types to which the same object might belong, and which have no particular relationship to each other. Examples abound: Emplyee, Customer, Dependent, Stockholder, parent, Patient, Pilot, Passenger, etc. Since a person might be an instance of any cominatin of these *n* types, this requires approximately 2^n subtypes of the form Employee_Customer, Customer_Dependent, Employee_Customer_Dependent, and so on.
>
> This proliferation is annoying and time-and space-consuming because:
>
> - There are so many of them.

- They introduce no new interface, since no new members will be defined. These types inherit everything from their supertypes. (In a sense, they are the converse of abstract types. They have instances but no members.)

- Changing types becomes a nightmare. Instead of simply asserting that an object has become a pilot, the user has to discover the object's current most specific type (say X_Y_Z), remove the object from X_Y_Z, and insert the object into X_Y_Z_Pilot (after determining the correct name of this type, which might also have been Pilot_X_Y_Z or X_Pilot_Y_Z and so on).

- Extending the schema by introducing a new type, say Consultant, requires the introduction of 2^n new types of the form X_Y_Z_Consultant.

Supporting multiple most specific types does pose language and implementation difficulties, but they are all solvable. Not providing such support simply shifts the problems to the user. Supporting multiple most specific types would address both of these problems.

334 The following Language Opportunity has been noted by X3H2-93-486/MUN-120 (edited by the Editor):

It would be useful to have an <add attributes definition> capability, probably in the context of general type evolution.

337 The following Language Opportunity has been noted by X3H2-93-488/MUN-122:

The concept *known not nullable* is not very useful when applied to attributes of ADTs.

341 The following Language Opportunity has been noted by X3H2-93-488/MUN-122:

It is a language opportunity to make <case expression>s and <case statement>s support multi-valued nulls.

342 The following Language Opportunity has been noted by X3H2-93-488/MUN-122:

<trigger definition>, clause 11.42 used to support a "statement list" but recently this was replaced by "<SQL procedure statement>". The logic was that the statement could be a <compound statement>. I agree with that but it is then a language opportunity to apply the same logic to <case statement> (SQL/PSM:clause 13.9), <if statement> (PSM:clause 13.10) <loop statement> (PSM clause 13.12), and <for statement> (PSM:clause 13.13). Whatever the eventual decision, we should be consistent throughout the document.

397 The following Language Opportunity has been noted by X3H2-94-293/SOU-146:

Component reference is currently not allowed as the target of the SET clause of an UPDATE statement.

417 The following Language Opportunity has been noted by X3H2-94-092/SOU-072:

<update by moving> in the <update statement: positioned> could be extended to apply to derived tables, and to supertables and subtables. Triggers could be introduced with a trigger event UPDATE MOVE.

4 SQL3 Language Features Not Approved by Both ANSI and ISO

In this section, I list (in no particular priority) the differences I observe in ANSI- and ISO-approved SQL3 language features.

4.1 Standing differences without recent papers to reconsider

In this first list, I include differences that were caused by papers either not approved or not considered at the most recent meeting(s) of ANSI and/or ISO.

None.

4.2 Standing differences with recent papers

In this second list, I include those differences that remain outstanding even though papers were considered at the most recent meeting(s) of ANSI and/or ISO, but that were not accepted or that did not completely resolve the differences.

1) Changes made by OTT-009/X3H2-92-055, *LIST, SET, and MULTISET as Type Templates*, as this paper was rejected by ISO.

2) X3H2-92-152, *Minutes of DBL RG*, KAW-084, *Updatability semantics*.

3) X3H2-92-152, *Minutes of DBL RG*, OTT-017, *Enhancement to updatability*.

4.3 Recent differences

In this next list, I include those differences resulting from the most recent meeting(s) of ANSI and /or ISO, which the other body has not yet had the opportunity to consider.

1) Subclause 4.21.1, "Checking of constraints", has differences resulting from X3H2-93-257/YOK-178/MUN-045.

2) Subclause 11.45, "<trigger definition>", has differences resulting from X3H2-93-258/YOK-179 /MUN-046.

3) Subclause 5.2, "<token> and <separator>", has differences from ISO's rejection of the <attributes function> in part 2.1 and all of part 2.5 in X3H2-93-109R/YOK-074/MUN-062.

4) Others, but I ran out of time for documenting them. Apologies!

5 Guidelines for writing "user-friendly" change proposals

This chapter of the Editor's Notes offers guidelines to the style of the content of the SQL document and all its Parts, as well as some guidelines for writing change proposals.

The Editor reserves the right to reject any change proposals that do not follow these guidelines, as appropriate.

5.1 Style of content of SQL document

5.1.1 General

5.1.1.1 Language

Since the SQL standard will be translated into other languages, and must in any case be read and understood by many whose first language is not English, simplicity of language is much more important than literary elegance.

Normal rules of good style apply; if in doubt, there are several excellent references to writing style that may be consulted.

The following specific points are noted:

— Conditions: The standard form used is "If *some condition* is satisfied, then *something happens*." This is strongly preferred to "*X* happens if *Y* is true." The "otherwise" case always deserves consideration, though it doesn't always need to be stated (particularly when the otherwise case involves no action or behavior).

— Number: It is clearer to stick to the singular and say, for example, "every *X* is destroyed" than to say "all *x*s are destroyed". Where "each" is acceptable, it is even better.

— Avoid the use of the word "any" as far as practicable. In ordinary English, it sometimes means "some" (as in "have you any shares in Company X?"), and sometimes means "every" (as in "you can call any day, any time").

  In particular, avoid constructions like "Let *CD* be any collation descriptor that includes . . . ". Use precise quantification, such as "For every collation descriptor *CD* that includes . . . ", and note that it is not necessary to add "if any"—even if there is no such collation descriptor, the rule still works fine.

— "which" versus "that": See X3H2-88-036 (copies available upon request). "That" should be used much more often than many people do.

5.1.1.2 Vocabulary

English is well known for having many synonyms (change, alter, amend), partially overlapping synonyms (describe and define, change and improve), and near-synonyms with subtle differences (edible and eatable), or even ignored distinctions (expect and anticipate). It may be assumed that any reader has a dictionary available, but where a word can be used in more than one sense, the correct interpretation should not depend on context any more than is absolutely necessary. If a word can have the same meaning or translation into another language as some other word used in SQL3 and no distinction is intended, then that other word should be used in preference.

Particular words used in specific senses in SQL3 include:

— **Definition**: Either a BNF non-terminal that causes the creation of something, or a definition of the meaning of a term.

— **Descriptor**: A persistent, formal description of an SQL object. See Subclause 3.3.5, "Descriptors", in Part 1.

— **Specify**: To state explicitly, for example “If GLOBAL is specified, . . . ” means “If the word ’GLOBAL’ actually appears, . . . ”. Note that “ . . . is specified . . . ” appears more frequently than “ . . . was specified . . . ”.

— **Case**: is favored. Nested cases are preferred to complex ones. Only the first rule is applied for which the condition is satisfied, so the *order* of the subrules is important. A Case construct normally ends with an “Otherwise” subrule, but this is not necessarily required.

Other terms worth noting are:

— **Database**: Note: this is one word. A collection of SQL-data. The term is to be avoided, because it raises the question of whether databases can overlap, be nested, are each described by a catalog or cluster of catalogs, *etc*. Usually, the term “SQL-data” is sufficient.

— **Description**: An informal description; not used in a technical sense. Note that the object that serves as a persistent description of an SQL object is known as a *descriptor*.

5.1.1.3 Quantification

When universally quantifying, prefer the singular “every” to the plural “all”—it nearly always works better. Although mathematicians often say “for all *x*”, it is not really good grammar (because the upside-down A (∀) of predicate calculus is considered to stand for “all”, the first letter of “every” having been taken by the backwards E (∃) that stands for “exists”).

When existentially quantifying, use “some”, not only in preference to “any”, but also sometimes in preference to the indefinite article “a” or “an”, which can be as ambiguous as “any”. However, this suggestion can lead to “overkill” text, so we further suggest that the indefinite article is appropriate in cases where it is clear that there can be no more than one occurrence of the thing in question, as in : “For every collation descriptor that includes a <translation collation> . . . ” (a collation descriptor includes at most one <translation collation>, so the word “some” instead of “a” could even be misleading here).

5.1.1.4 Quotation marks

Single <quote>s enclose <character string literal>s; <double quote>s enclose <delimited identifier>s. In text, double quotes are normally used to surround quoted material.

5.1.1.5 Typography

— <key word>s (only) are in <simple Latin upper case letter>s.

— Truth values (as opposed to Boolean values) are lower-case, italicized, and underlined: *true* , *false* , and *unknown* .

— With the exception of certain index entries and certain table headings, the only bolding in the document is of:

- Ada keywords such as **package**, because the Ada convention is followed;
- The terms being defined in Subclause 3.1, "Definitions"; and
- Names of pseudo-procedures (visible now only in the definition of significant DB_changes, where the procedure **proc_VC** is defined).

— Clauses ("chapters" to some) are the major divisions in the document; Subclauses are all the lesser divisions. The names of Clauses and Subclauses are all spelled with initial capital letters; subsequent words in those names are spelled with initial lower-case letters—except, of course, for such words as "SQL".

5.1.2 SQL terminology

5.1.2.1 Terms for SQL objects

Terms referring to SQL objects must be carefully chosen, and used not only correctly but wherever possible.

Such terms are frequently prefixed with "SQL-" in order to distinguish them from similar terms used with different meanings in related contexts. For example: SQL-transaction, SQL-session, SQL-client. Note that SQL-schema is, strictly, the correct term, because it has a different meaning from "schema" in the Reference Model of Data Management, but "schema" is used throughout the SQL document because it is always the SQL sense that is intended.

Ambiguities such as "object identifier" should be avoided.

5.1.2.2 BNF non-terminals

The names of BNF non-terminals do not often contain hyphens; where they do, it is normally because the analogous English phrase would contain hyphens ("form-of-use", "implementation-defined"). They sometimes contain colons to distinguish closely related constructs (for example, <delete statement: positioned> and <delete statement: searched>).

Care should be taken not to introduce over-general names. For example, <SQL statement> is unfortunate in that its production rule does not allow any SQL-statement.

Every new <key word> must be specified as a <reserved word> or <non-reserved word>, as appropriate.

Precision in referring to BNF non-terminals is important. For example: there is no such thing as a <statement>.

Do not fall into the trap of assuming that the name of a BNF non-terminal means the same thing as the words contained within the angle brackets. When writing a rule about <select list>, do not then start using the phrase "select list" as though it has some meaning. BNF non-terminals are nothing more than distinguished character strings whose spelling is irrelevant except for consistent use. If "<select list>" were replaced "automatically" with "<SL>", one would not feel comfortable discussing the "SL" as a normal English phrase.

5.1.2.3 BNF non-terminal or name of SQL object?

BNF non-terminals do not denote persistent objects. In particular, it is incorrect to say "If an <assertion definition> exists . . . "; an <assertion definition> is source code, which, once processed, ceases to exist as far as the SQL implementation is concerned. The result of processing it is an *assertion*, represented by an *assertion descriptor* in some schema.

data type or <data type>? "data type" is the correct term, unless it is specifically intended to refer to the BNF nonterminal "<data type>" (probably appropriate only when discussing the syntax).

Where reference is intended to a generic data type, such a character string type, then that is the preferred term.. SQL, at the time of writing, uses "character data type", "<character string type>", "data type CHARACTER", and several other phrases almost synonymously.

"<preparable statement>" is a BNF non-terminal; "prepared statement" is not.

5.1.2.4 Notes on specific SQL terms

"contain" is used for syntactic containment, *i.e.*, one BNF non-terminal is said to contain others. "data type" is two words. "datetime" is one word. "include" is used for specifying that one descriptor is included in others.

5.1.2.5 Locally-defined terms and symbols

Diacritical or other marks, such as prime, are to be avoided for typographical reasons.

Other considerations involving terms and symbols:

— Symbolic names are italicized and composed of <simple Latin upper case letter>s and <digit>s. Examples: *T*, *C1*, and *SLCC*.

— Indexes or subscripts are in lower-case italic letters and digits. Examples: "the *i*-th column", "C_k", or "at least *j* values".

— Special symbols for use as "range variables" (like the "CD" earlier) should not be introduced unnecessarily. For example:

> Every aardvark-reference *A* that contains . . . is removed from every zoological paper that includes *A*.

is preferable to:

> For every zoological paper *ZP*, for every aardvark-reference *A* that contains . . . and is included in *ZP*, *A* is removed from *ZP*.

Special symbols are necessary to avoid clumsiness and to ensure precision in many of the more complicated rules.

These special symbols' scope is the Subclause in which they are defined. While they may be defined early in a Subclause (for example, in the Syntax Rules) and used much later (for example, in the General Rules), it is preferable to define them in the section in which they are used within a Subclause.

5.1.2.6 Abbreviations

Abbreviations such as "auth-id" are not used in the SQL standard, however widely they may be generally used and understood. It follows that they are not acceptable in text proposed for insertion in the document.

5.1.3 The content of the SQL standard

5.1.3.1 Definitions

This section should conform to the (normative!) Annex B.1 of the ISO Directives, Part 3.

A term should be defined in this section if and only if:

— It is used through the Standard, rather than in any particular context; and

— Either:

- It is a term not widely used or that has been invented for the Standard; or
- It has more than one meaning and a precise meaning is consistently intended in the Standard.

A term should not be included simply for tutorial purposes (the Standard is already large enough).

5.1.3.2 Concepts

Although Clause 4, "Concepts", was originally introduced purely for the purpose of explanation, it now contains much material that is definitive. There are no guidelines to what should or should not be there. Thus Subclause 4.2.3, "Rules determining collating sequence usage", are classed as concepts, while tables of valid casts and valid values of datetime data types are not.

5.1.3.3 Modularity

Except perhaps where the result would be a ridiculously short Subclause, the specification of a BNF non-terminal used in more than one place should be in a separate Subclause from every one of its uses. Otherwise, there is a serious risk of unintended effects resulting from changing the Rules in one place without realizing that they should remain unchanged for the use elsewhere.

In particular, the construct “Every General Rule except General Rule 1 of Subclause x.y applies here with ’something’ replaced by ’something else’” is to be avoided at all costs.

5.1.3.4 Format rules

The syntax of the version of BNF used is defined in Subclause 3.2, "Notation".

Every SQL character that is not a letter or digit has a name, defined in Subclause 5.1, "<SQL terminal character>". Except in those BNF productions that define names for them, such characters never appear standing for themselves, but are always represented by their names. Thus, except where they are specified as <left bracket>, *etc.*, the characters “[”, “]”, “{”, “}”, “ | ”, and “ . . . ” in BNF productions always serve as "punctuation".

Therefore, the production:

```
<x comma list> ::= ( <x> [ , <x> ]... )
```

should always be written as:

```
<x comma list> ::= <left paren> <x> [ <comma> <x> ]... <right paren>
```

5.1.3.5 Syntax and Access Rules, and General Rules

The differences between Syntax and Access Rules on the one hand and General Rules on the other are that Syntax and Access Rules specify requirements for a program to conform to (some level of) the SQL standard, while General Rules specify what a standard-conforming SQL implementation is required to do when it process such a standard-conforming program.

Where there are no Rules of any kind, it is usual to say “None.”, to exclude the possibility that they have not been thought about, or have been lost.

5.1.3.6 Syntax Rules

The correct form is “This condition shall be satisfied . . . ” or “If *X* is specified, then *Y* shall be specified”.

Syntax Rules *do not* deal with the privileges required to accomplish some action.

5.1.3.7 Access Rules

The correct form is as for Syntax Rules.

Access Rules *do* deal with the privileges required to accomplish some action.

5.1.3.8 General Rules

General Rules specify the effect of processing SQL source code that satisfies the relevant Format, Syntax Rules, and Access Rules. The preferred form is present indicative passive, *e.g.*, “A descriptor is created . . . ”.

By the time the Syntax and Access Rules have been processed, certain implications have become explicit. Thus, if the optional <schema name> of a <table name> is omitted, it is deduced, and by the time General Rule 1) is encountered, it can be assumed that <schema name> is known. It is not normally necessary or desirable to mention that it is “explicit or implicit”.

5.1.3.9 Exceptions

There are two categories of end-of-statement behavior:

— “ . . . an exception condition is raised: *some condition*”; and

— “ . . . a completion condition is raised: *limited alternatives*”. The only acceptable alternatives for the completion conditions are *successful completion*, *warning*, and *no data*. Each of these may have no subcode or a specific subcode.

The exception class and subclass phrases are separated by a dash “—” and are always both in italics.

5.1.3.10 Leveling Rules

The correct form is as for Syntax Rules. A Rule that illustrates a point by the use of an SQL expression either shows the SQL expression on a separate line:

Let *B* be an exact numeric result of the operation:

CAST (CAST (*Y* AS *Q*) AS *E2*)

or encloses the expression in double-quotes:

“*R* is NULL” is true if and only if . . .

5.2 Writing change proposals

5.2.1 Discussion

It is helpful to the reader if the discussion part makes quite clear why the proposal is being made. It also helps if any approaches that were considered and rejected are also mentioned, together with the reasons for their rejection.

Proposal writers should strive to describe *all* changes and their implications in the discussion part. Readers should have have to slog through the detailed language changes to understand the consequences of the change.

5.2.2 Change proposals

A) Every separate item of the proposal should be numbered.

B) When referring to a Subclause of the document, always specify both Subclause number and the name of the Subclause (for example, Subclause 6.1, "<data type>").

C) When inserting, deleting, or replacing text or Rules in the document, specify both the reference, *e.g.*, Syntax Rule 11)c)iv)2)B), and sufficient context to identify the specific paragraph, sentence, or Rule to be changed.

For example, to replace Subclause x.y, Syntax Rule 11)c)iv)2)B), each of Syntax Rules 11), 11)c), 11)c)iv), and 11)c)iv)2) should be unambiguously defined. Where, as for example in Subclause 13.9, "<using clause>", several Rules start off very similarly, say something like "Replace Syntax Rule 11)c)iv)2)B) (the Rule that deals with the *xyz* in an *rst*) with...".

D) Where only a few words of existing text are to be changed, it is clearer both to the reviewer and the Editor to flag differences in some way, *e.g.*, by striking out deleted text and underlining new. Whatever convention is used should be explained.

5.2.3 Check list

All changes to the document must be specified explicitly. It is unfair to the Editor to expect him to make changes left implicit.

The following list is provided in the hope of reducing the number of incomplete change proposals:

- Leveling Rules and Appendix A, "Leveling the SQL Language",
- Appendix B, "Implementation-defined elements" or Appendix C, "Implementation-dependent elements",
- Appendix E, "Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992", especially for new <reserved word>s,
- Appendix E, "Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992", for new class or subclass values—also please remind the Editor to index the new SQLSTATE values; also update the table of Ada package values in SQL/Bindings, Subclause 10.2, "<routine>".
- The Dynamic SQL Descriptor Area
- Information and Definition Schemas.

A proposal that solves a Possible Problem, Minor Problem, or Opportunity, whether intentionally or not, should say so prominently, so that the Editor's Notes may be kept up-to-date.

A proposal that introduces a Possible Problem, for example because it is admittedly incomplete, should specify an appropriate addition to the Possible Problems list. The same applies to language opportunities.

A proposal to insert lengthy pieces of text into the document should, either before or as soon as practicable after it has been adopted, be provided to the Editor in machine-readable form, either on diskette or by electronic mail (see Clause 6, "Machine readable change proposals" for more information).

It is accepted that this machine-readable version will be in "plain text" form and without many formatting attributes (*e.g.*, bolding, underlining, font changes), but it is nevertheless of great help to the Editor.

Because of the increasing size and partitioning of the SQL3 document and the great complexity of dealing with proposals that make substantial changes to the document's structure and content, I reserve the right to decline instructions to change the document when those instructions are not complete and that do not generally follow these guidelines.

Specifically, if a change proposal is accepted that fails to cite the number and title of the document being changed by the proposal, the proper Clause and/or Subclause numbers and titles affected by the proposal, and the Rule numbers and partial text of Rules that are changed by the proposal, then I will not make any changes in the next edition of the base document, but will bring the paper back to the Committee for revision and completion. Thanks!

6 Machine readable change proposals

I would like to thank those ISO and ANSI participants who have provided me with copies of change proposals on diskettes. This practice reduces the editorial workload considerably and makes it much easier to ensure accurate reflection of the change proposals in the base document. I continue to urge those participants who can supply diskette copies of lengthier proposals (i.e., a page or more of new text) to bring those diskettes to meetings, in order to avoid mail delays. I will of course either return the diskettes or replace them with blank diskettes on request.

I prefer 3.5 inch or 5.25 inch DS/DD soft-sectored diskettes, with DOS files (clearly labeled in any case!). I may be able to find ways to copy other types of diskette and file, but would prefer to avoid the hassles if possible. Please include both unformatted and formatted versions of each file, if possible. Fully-justified text (with the extra blanks inserted for alignment purposes) do not appear to be a problem for my document processor.

Alternatively, it may be possible for you to electronically mail me your change proposals. My "generic" Internet mail address is:

melton@cookie.enet.dec.com

I am told that wizards having sufficient skills should be able to translate these addresses into the appropriate incantations necessary for transporting text files to me. Good luck...

Index

Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF nonterminal was defined; index entries appearing in *italics* indicate a page where the BNF nonterminal was used in a Format; and index entries appearing in roman type indicate a page where the word, phrase, or BNF nonterminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Leveling Rule, Table, or other descriptive text.

272 • 18
273 • 18, 22
274 • 18
275 • 42
278 • 18
279 • 19
281 • 42
282 • 19
283 • 19
284 • 19
285 • 19
286 • 19
287 • 19
288 • 19
289 • 19
290 • 19
291 • 42
292 • 20
293 • 20
294 • 20
295 • 20
296 • 21
297 • 21
299 • 21
301 • 21
302 • 21
303 • 21
304 • 21
305 • 22
306 • 22
307 • 22
308 • 22
309 • 42
310 • 22
311 • 43
312 • 23
315 • 23
316 • 12
317 • 43
318 • 23
319 • 23
320 • 23
321 • 23
322 • 23
323 • 23
324 • 24
325 • 24
326 • 24
327 • 43
328 • 43
329 • 24
330 • 24
331 • 25
332 • 25
333 • 25
334 • 44
335 • 26
336 • 26
337 • 44
338 • 26
339 • 26
340 • 26
341 • 44
342 • 44
343 • 26
344 • 27
346 • 27
347 • 27
348 • 27
349 • 27
350 • 27
351 • 27
352 • 28
353 • 28
354 • 28
355 • 28
356 • 28
357 • 28
358 • 29
359 • 29
360 • 29
361 • 29
362 • 29
363 • 30
364 • 30
365 • 30
366 • 30
367 • 30
368 • 30
369 • 30
370 • 30, 31
371 • 30
372 • 30
373 • 30
374 • 31
375 • 31
376 • 31
377 • 31
378 • 31
379 • 31
380 • 31
381 • 31
382 • 31
383 • 31
384 • 32
385 • 32
386 • 32
387 • 32
389 • 32
390 • 32
391 • 33
392 • 33
393 • 33
394 • 33
395 • 33
396 • 33
397 • 44
398 • 34

— P —

— S —

ISO-ANSI Working Draft
Database Language SQL/Foundation (SQL3)
«Part 2»

August 1994

Contents

Page

TABLES

Table **Page**

Foreword

ISO Only–SQL3

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at leat 75% of the national bodies casting a vote.

International Standard ISO/IEC 9075 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*.

This fourth edition cancels and replaces the third edition (ISO/IEC 9075:1992).

Annexes A, B, C, D, E, and F of this International Standard are for information only.

ANSI Only–SQL3

This Standard (American National Standard X3.135-199*x*, *Database Language—SQL*), is a revision of American National Standard X3.135-1992, (*Database Language*), that adds significant new features and capabilities to the specifications.

ANSI (the American National Standards Institute) is the United States national standards body charged with development of American National Standards.

This Standard was approved as an American National Standard by the American National Standards Institute on (**insert date here**).

Suggestions for improvement of this Standard are welcome. They should be sent to the Computer and Business Equipment Manufacturing Assocation, 311 First Street, NW, Washington, DC 20011.

This Standard was processed and approved for submittal to ANSI by the Accredited National Standards Committee on Information Processing Systems, X3. Committee approval of this Standard does not necessarily imply that all committee members voted for its approval. At the time that it approved this Standard, the X3 Committee had the following members.

(**Insert X3 officer and membership lists here.**)

American National Standard X3.135-199*x* was prepared by Technical Committee Group X3H2, Database Languages, working under the auspices of Accredited National Standards Committee X3, Information Processing Systems. Technical Committee X3H2 on Database, which developed this Standard, had the following members:

Donald R. Deutsch, Chair
Bruce M. Horowitz, Vice-Chair
Michael M. Gorman, Secretary
Leonard J. Gallagher, International Representative

(**Insert membership list here.**)

Others holding Technical Committee X3H2 membership while the committee was developing this Standard are the following:

(**Insert other membership list here.**)

This American Standard was approved in 199*x*.

This American Standard was developed from ANS X3.135-1992, Database Language SQL, and replaces that American Standard. It adds significant new features and capabilities to the specifications. It is generally compatible with ANS X3.135-1992 in the sense that, with very few exceptions, SQL language that conforms to ANS X3.135-1992 also conforms to this American Standard, and will be treated in the same way by an implementation of this American Standard as it would by an implementation of ANS X3.135-1992 The known incompatibilities between ANS X3.135-1992 and this American Standard are stated in Informative Annex E, "Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992".

Technical changes between ANS X3.135-1992 and this American Standard include both improvements or enhancements to existing features and the definition of new features. Significant improvements in existing features include:

To Be Supplied.

Significant new features are:

1) Support for active “rules”, called triggers.

2) Support for abstract data types.

3) Support for multiple null states.

4) Support for PENDANT referential integrity.

5) A recursive union operation for query expressions.

6) Support for enumerated and boolean data types.

7) Support for SENSITIVE cursors.

This American Standard contains six Informative annexes:

— Annex A (informative): Leveling the SQL Language;

— Annex B (informative): Implementation-defined elements;

— Annex C (informative): Implementation-dependent elements;

— Annex D (informative): Deprecated features;

— Annex E (informative): Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992;

— Annex F (informative): Maintenance and interpretation of SQL.

Introduction

The organization of this
ANSI American
ISO International
Standard is as follows:

1) Clause 1, "Scope", specifies the scope of this
ANSI American
ISO International
Standard.

2) Clause 2, "Normative references", identifies additional standards that, through reference in this
ANSI American
ISO International
Standard, constitute provisions of this
ANSI American
ISO International
Standard.

3) Clause 3, "Definitions, notations, and conventions", defines the notations and conventions used in this
ANSI American
ISO International
Standard.

4) Clause 4, "Concepts", presents concepts used in the definition of SQL.

5) Clause 5, "Lexical elements", defines the lexical elements of the language.

6) Clause 6, "Scalar expressions", defines the elements of the language that produce scalar values.

7) Clause 7, "Query expressions", defines the elements of the language that produce rows and tables of data.

8) Clause 8, "Predicates", defines the predicates of the language.

9) Clause 9, "Data assignment rules and function determination", specifies the rules for assignments that retrieve data from or store data into the database, and formation rules for set operations.

10) Clause 10, "Additional common elements", defines additional language elements that are used in various parts of the language.

11) Clause 11, "Schema definition and manipulation", defines facilities for creating and managing a schema.

12) Clause 12, "Module", defines modules and procedures.

13) Clause 13, "Data manipulation", defines the data manipulation statements.

14) Clause 14, "Transaction management", defines the SQL-transaction management statements.

15) Clause 15, "Connection management" defines the SQL-connection management statements.

16) Clause 16, "Session management", defines the SQL-session management statements.

17) Clause 17, "Diagnostics management", defines the diagnostics management facilities.

18) Clause 18, "Information Schema and Definition Schema", defines viewed tables that contain schema information.

19) Clause 19, "Status codes", defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

20) Clause 20, "Conformance", defines the criteria for conformance to this
ANSI American
ISO International
standard.

21) Annex A, "Leveling the SQL Language", is an informative Annex. It lists the leveling rules defining the Entry SQL and Intermediate SQL subset levels SQL levels of the SQL language.

22) Annex B, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of the
ANSI American
ISO International
Standard states that the syntax or meaning or effect on the database is partly or wholly implementation-defined, and describes the defining information that an implementor shall provide in each case.

23) Annex C, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of the
ANSI American
ISO International
Standard states explicitly that the meaning or effect on the database is implementation-dependent.

24) Annex D, "Deprecated features", is an informative Annex. It lists features that the responsible Technical Committee intend will not appear in a future revised version of this
ANSI American
ISO International
Standard.

25) Annex E, "Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992", is an informative Annex. It lists incompatibilities with the previous version of this
ANSI American
ISO International
Standard.

26) Annex F, "Maintenance and interpretation of SQL", is an informative Annex. It identifies SQL interpretations and corrections that have been processed by
ANSI ANSI Accredited Committee X3
ISO ISO/IEC JTC1/SC21
since adoption of
ANSI ANSI X3.135-1989.

ISO ISO/IEC 9075:1989.

In the text of this
ANSI American
ISO International
Standard, Clauses begin a new odd-numbered page, and in Clause 5, "Lexical elements", through Clause 19, "Status codes", Subclauses begin a new page. Any resulting blank space is not significant.

Information Technology — Database Languages — SQL

1 Scope

This
ANSI American
ISO International
Standard defines the data structures and basic operations on SQL-data. It provides functional capabilities for creating, accessing, maintaining, controlling, and protecting SQL-data.

Note: The framework for this
ANSI American
ISO International
Standard is described by the Reference Model of Data Management (ISO DIS 10032:1991).

This
ANSI American
ISO International
Standard specifies the syntax and semantics of a database language:

— for specifying and modifying the structure and the integrity constraints of SQL-data;

— for declaring and invoking operations on SQL-data and cursors; and

— for declaring database language procedures.

It also specifies an Information Schema that describes the structure and the integrity constraints of SQL-data.

This
ANSI American
ISO International
Standard provides a vehicle for portability of data definitions and compilation units between SQL-implementations.

This
ANSI American
ISO International
Standard provides a vehicle for interconnection of SQL-implementations.

This
ANSI American
ISO International
Standard does not define the method or time of binding between any of:

— database management system components,

— SQL data definition declarations,

— SQL procedures, or

— compilation units.

Implementations of this
ANSI American
ISO International
Standard may exist in environments that also support application programming languages, end-user query languages, report generator systems, data dictionary systems, program library systems, and distributed communication systems, as well as various tools for database design, data administration, and performance optimization.

2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this
ANSI American
ISO International
Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this
ANSI American
ISO International
Standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ANSI Only–SQL3

— ANSI X3.9-1978, *American National Standard Programming Language FORTRAN.*

— ANSI X3.23-1985, *American National Standard for Information Systems—Programming Language—COBOL.*

— ANSI X3.53-1976, *American National Standard Programming Language PL/I.*

— ANSI X3.159-1989, *American National Standard for Information Systems—Programming Language—C.*

— ANSI X3.198-1991, *American National Standard for Information Systems—Programming Language—Fortran.*

Note: ANSI X3.198-1991 introduces no incompatibilities with ANSI X3.9-1978 that affect the binding between Fortran and SQL; therefore, wherever "Fortran" is specified in this
ANSI American
ISO International
Standard, ANSI X3.198-1991 is implicit.

— ANSI/MDC X11.1-1990, *American National Standard for Information Systems—Programming Language—MUMPS.*

— ANSI/IEEE 770/X3.97-1983, *American National Standard for Information Systems—Programming Language—Pascal.*

— ANSI/IEEE 770/X3.160-1989, *American National Standard for Information Systems—Programming Language—Extended Pascal.*

— ANSI/MIL-STD-1815A-1983, *American National Standard for Information Systems—Reference Manual for the Ada® Programming Language.*

— ISO/IEC 646:1991, *Information technology—ISO 7-bit coded character set for information interchange.*

— ISO 2022:1986, *Information technology—ISO 7-bit and 8-bit coded character sets—code extension techniques.*

— ISO 8601:1988, *Data elements and interchange formats – Information interchange—Representation of dates and times.*

— ISO 8649:1988, *Information Processing Systems—Open Systems Interconnection—Service Definition for the Association Control Service Element*

— ISO/IEC 8824:1990, *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1).*

— ISO/IEC 9579-1:1993(E), *Information Technology—Open Systems Interconnection—Remote Database Access, Part 1: Generic Model, Service, and Protocol*

— ISO/IEC 9579-2:1993(E), *Information technology—Open Systems Interconnection — Remote Database Access, Part 2: SQL specialization.*

— ISO/IEC 10026-2, *Information Technology—Open Systems Interconnection—Distributed Transaction Processing - Part 2: Service Definition*

— ISO/IEC 10646:1993, *Information technology – Multiple-octet coded character set.*

— ISO/IEC CD 11404, *Common Language-Independent Datatypes (CLID).*

ISO Only–SQL3

— ISO/IEC 646:1991, *Information technology—ISO 7-bit coded character set for information interchange.*

— ISO/IEC 1539:1991, *Information technology—Programming languages—Fortran.*

— ISO 1989:1985, *Programming languages—COBOL.*
(Endorsement of ANSI X3.23-1985).

— ISO 2022:1986, *Information technology—ISO 7-bit and 8-bit coded character sets—code extension techniques.*

— ISO 6160:1979, *Programming languages—PL/I*
(Endorsement of ANSI X3.53-1976).

— ISO/IEC 7185:1990, *Information technology—Programming languages—Pascal.*

— ISO 8601:1988, *Data elements and interchange formats – Information interchange—Representation of dates and times.*

— ISO 8649:1988, *Information Processing Systems—Open Systems Interconnection—Service Definition for the Association Control Service Element*

— ISO 8652:1987, *Programming languages—Ada.*
(Endorsement of ANSI/MIL-STD-1815A-1983).

— ISO/IEC 8824:1990, *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1).*

— ISO/IEC 9579-1:1993(E), *Information Technology—Open Systems Interconnection—Remote Database Access, Part 1: Generic Model, Service, and Protocol*

— ISO/IEC 9579-2:1993(E), *Information technology—Open Systems Interconnection — Remote Database Access, Part 2: SQL specialization*.

— ISO/IEC 9899:1990, *Programming languages – C*.

— ISO/IEC 10026-2, *Information Technology—Open Systems Interconnection—Distributed Transaction Processing - Part 2: Service Definition*

— ISO/IEC 10206:1991, *Information technology—Programming languages—Extended Pascal*.

— ISO/IEC 10646:1993, *Information technology – Multiple-octet coded character set*.

— ISO/IEC CD 11404, *Common Language-Independent Datatypes (CLID)*.

— ISO/IEC 11756:1992, *Information technology—Programming languages—MUMPS*.

More to be supplied as required

3 Definitions, notations, and conventions

3.1 Definitions

For the purposes of this
ANSI American
ISO International
Standard, the following definitions apply.

3.1.1 Definitions taken from ISO/IEC DIS 10646

This
ANSI American
ISO International
Standard makes use of the following terms defined in ISO/IEC 10646:

a) **character**

b) **octet**

c) **variable-length coding**

d) **fixed-length coding**

3.1.2 Definitions taken from ISO 8601

This
ANSI American
ISO International
Standard makes use of the following terms defined in ISO 8601:

a) **Coordinated Universal Time** (UTC)

b) **date** ("date, calendar" in ISO 8601)

3.1.3 Definitions provided in this Standard

This
ANSI American
ISO International
Standard defines the following terms:

a) **abstract data type—ADT**: Specification of behavior of similar objects, called *instances* of that ADT. An important aspect of ADTs is the separation of the *interface* of the type from its *implementation*.

b) **actor functions**: Functions that are either observer functions or mutator functions.

c) **assignable**: The characteristic of a value or of a data type that permits that value or the values of that data type to be assigned to data instances of a specified data type. See Subclause 4.15, "Type conversions and mixing of data types".

d) **attribute**: An attribute is a stored component (of an ADT) that has a name and a data type.

e) **cardinality** (of a collection): The number of objects in that collection. Those objects need not necessarily have distinct values.

f) **character repertoire**; **repertoire**: A set of characters used for a specific purpose or application. Each character repertoire has an implied default collating sequence.

g) **coercibility**: An attribute of character string data items that governs how a collating sequence for the items is determined.

h) **collation**; **collating sequence**: A method of ordering two comparable character strings. Every character set has a default collation.

i) **comparable**: The characteristic of two data objects that permits the value of one object to be compared with the value of the other object. Also said of data types: Two data types are comparable if objects of those data types are comparable. If one of the two data types is an abstract data type, then both shall be in the same subtype family. See Subclause 4.15, "Type conversions and mixing of data types".

j) **component (of an ADT)**: One element of the representation.

k) **constructor operations**: Operations that create new instances of an ADT as SQL-data.

l) **destructor functions**: Functions that remove SQL-data that represents an instance of an ADT.

m) **descriptor**: A coded description of an SQL object. It includes all of the information about the object that a conforming SQL-implementation requires.

n) **distinct**: Two values are said to be not distinct if either: both are the null value, or they compare equal according to Subclause 8.2, "<comparison predicate>". Otherwise they are distinct. Two rows (or partial rows) are distinct if at least one of their pairs of respective values is distinct. Otherwise they are not distinct. The result of evaluating whether or not two values or two rows are distinct is never unknown.

o) **duplicate**: Two or more values or rows are said to be duplicates (of each other) if and only if they are not distinct.

p) **dyadic operator**: An operator having two operands: a left operand and a right operand. An example of a dyadic arithmetic operator in this
ANSI American
ISO International
Standard is "–", specifying the subtraction of the right operand from the left operand.

q) **encapsulation**: A mechanism to limit visibility of certain members of an ADT. Each member of an ADT is specified with an encapsulation level of either public, private, or protected. A private member is visible only within the definition of the ADT itself (including its operations), whereas a public member is also visible outside the ADT definition wherever the ADT itself is visible. A protected member is intermediate between these, being visible only within the definition of the ADT and the definition of any subtype of the ADT.

r) **form-of-use**: A convention (or encoding) for representing characters (in character strings). Some forms-of-use are fixed-length codings and others are variable-length codings.

s) **form-of-use conversion**: A method of converting character strings from one form-of-use to another form-of-use.

t) **generated type**: an ADT resulting from specifying the name of a type template and a set of template parameters required by that type template

u) **implementation (of an ADT)**: Some of the operations associated with an ADT might be realized by means of data that is stored as SQL-data, while other operations might be realized as executable code (functions) that implements the manipulations that can be performed on, to, or with, an object. Stored data together with the data structures and code that implement the behavior of an ADT is its *implementation*.

v) **implementation-defined**: Possibly differing between SQL-implementations, but specified by the implementor for each particular SQL-implementation.

w) **implementation-dependent**: Possibly differing between SQL-implementations, but not specified by this
ANSI American
ISO International
Standard and not required to be specified by the implementor for any particular SQL-implementations.

x) **interface (of an ADT)**: The behavior of an ADT, specified through a set of those attributes and operation signatures that are not encapsulated.

y) **list**: An ordered collection of
ANSI elements
ISO objects
that are not necessarily distinct. The collection may be empty.

z) **member**: An attribute or operation of an ADT.

aa) **monadic operator**: An operator having one operand. An example of a monadic arithmetic operator in this
ANSI American
ISO International
Standard is –, specifying the negation of the operand.

bb) **multiset**: An unordered collection of
ANSI elements
ISO objects
that are not necessarily distinct. The collection may be empty.

cc) **mutator functions**: Functions that, given an ADT instance and a new value, change some part of the state of the ADT instance.

dd) ***n*-adic operator**: An operator having a variable number of operands (informally: *n* operands). An example of an *n*-adic operator in this
ANSI American
ISO International
Standard is COALESCE.

ee) **null value (null)**: A special value, or mark, that is used to indicate the absence of any data value.

ff) **object**: an instance of an object ADT; each instance is identified by a unique object identifier.

gg) **object ADT**: an abstract data type defined with the OBJECT clause.

hh) **object identifier (OID)**: An immutable unique value associated with an ADT instance that is independent of the ADT value.

ii) **observer functions**: Functions that do not change the state of an ADT instance and, given the ADT instance, may return values derived from the value of an ADT instance.

jj) **persistent**: Continuing to exist indefinitely, until destroyed deliberately. Referential and cascaded actions are regarded as deliberate. Actions incidental to the ending of an SQL-transaction (see Subclause 4.39, "SQL-transactions") or an SQL-session (see Subclause 4.41, "SQL-sessions") are not regarded as deliberate.

kk) **redundant duplicates**: All except one of any multiset of duplicate values or rows.

ll) **repertoire**: See **character repertoire**.

mm) **representation (of an ADT)**: The ordered sequence of attributes of the ADT, including both those that are encapsulated and those that are not.

nn) **row**: The data type of a sequence of (<column name>, data type) pairs

---

ANSI Only–SQL3

---

oo) **row type**: The data type of a row of a table, described by a sequence of (<column name>, data type) pairs.

---

pp) **sequence**: An ordered collection of objects that are not necessarily distinct.

qq) **set**: An unordered collection of distinct
ANSI elements.
ISO objects.
The collection may be empty.

rr) **signature (of a routine)**: the name of a routine, the position and data types of each of its parameters, and an indication of whether it is a function or a procedure.

ss) **state (of an ADT instance)**: The ordered sequence of components of an ADT instance, excluding any component named OID.

tt) **template parameters**: A type template is defined with one or more formal template parameters. A generated type is denoted by a <generated type reference>, which consists of the name of a type template and a set of actual template parameters. Template parameters may be value template parameters or type template parameters.

uu) **translation**: A method of translating characters in one character repertoire into characters of the same or a different character repertoire.

vv) **type template**: the definition of a "meta type" that is used to specify a generated type.

ww) **type template parameter**: a template parameter used to specify a <data type>.

xx) **value (of an ADT instance)**: See *state*.

yy) **value ADT**: an abstract data type defined with the VALUE clause.

zz) **value template parameter**: a template parameter used to specify a value of some <data type>.

3.2 Notation

The syntactic notation used in this
ANSI American
ISO International
Standard is an extended version of BNF ("Backus Normal Form" or "Backus Naur Form").

In BNF, each syntactic element of the language is defined by means of a *production rule*. This defines the element in terms of a formula consisting of the characters, character strings, and syntactic elements that can be used to form an instance of it.

The version of BNF used in this
ANSI American
ISO International
Standard makes use of the following symbols:

Symbol	*Meaning*
< >	Angle brackets delimit character strings that are the names of syntactic elements, the non-terminal symbols of the SQL language.
::=	The definition operator. This is used in a production rule to separate the element defined by the rule from its definition. The element being defined appears to the left of the operator and the formula that defines the element appears to the right.
[ ]	Square brackets indicate optional elements in a formula. The portion of the formula within the brackets may be explicitly specified or may be omitted.
{ }	Braces group elements in a formula. The portion of the formula within the braces must be explicitly specified.
|	The alternative operator. The vertical bar indicates that the portion of the formula following the bar is an alternative to the portion preceding the bar. If the vertical bar appears at a position where it is not enclosed in braces or square brackets, it specifies a complete alternative for the element defined by the production rule. If the vertical bar appears in a portion of a formula enclosed in braces or square brackets, it specifies alternatives for the contents of the innermost pair of such braces or brackets.
. . .	The ellipsis indicates that the element to which it applies in a formula may be repeated any number of times. If the ellipsis appears immediately after a closing brace "}", then it applies to the portion of the formula enclosed between that closing brace and the corresponding opening brace "{". If an ellipsis appears after any other element, then it applies only to that element.
!!	Introduces normal English text. This is used when the definition of a syntactic element is not expressed in BNF.

Spaces are used to separate syntactic elements. Multiple spaces and line breaks are treated as a single space. Apart from those symbols to which special functions were given above, other characters and character strings in a formula stand for themselves. In addition, if the symbols to the right of the definition operator in a production consist entirely of BNF symbols, then those symbols stand for themselves and do not take on their special meaning.

Pairs of braces and square brackets may be nested to any depth, and the alternative operator may appear at any depth within such a nest.

A character string that forms an instance of any syntactic element may be generated from the BNF definition of that syntactic element by application of the following steps:

1) Select any one option from those defined in the right hand side of a production rule for the element, and replace the element with this option.

2) Replace each ellipsis and the object to which it applies with one or more instances of that object.

3) For every portion of the string enclosed in square brackets, either delete the brackets and their contents or change the brackets to braces.

4) For every portion of the string enclosed in braces, apply steps 1 through 5 to the substring between the braces, then remove the braces.

5) Apply steps 1 through 5 to any non-terminal syntactic element (i.e., name enclosed in angle brackets) that remains in the string.

The expansion or production is complete when no further non-terminal symbols remain in the character string.

3.3 Conventions

3.3.1 Informative elements

In several places in the body of this
ANSI American
ISO International
Standard, informative notes appear. For example:

Note: This is an example of a note.

Those notes do not belong to the normative part of this
ANSI American
ISO International
Standard and conformance to material specified in those notes shall not be claimed.

3.3.2 Specification of syntactic elements

Syntactic elements are specified in terms of:

— **Function**: A short statement of the purpose of the element.

— **Format**: A BNF definition of the syntax of the element.

— **Syntax Rules**: A specification of the syntactic properties of the element, or of additional syntactic constraints, not expressed in BNF, that the element shall satisfy, or both.

— **Access Rules**: A specification of rules governing the accessibility of schema objects that must hold before the General Rules may be successfully applied.

— **General Rules**: A specification of the run-time effect of the element. Where more than one General Rule is used to specify the effect of an element, the required effect is that which would be obtained by beginning with the first General Rule and applying the Rules in numeric sequence unless a Rule is applied that specifies or implies a change in sequence or termination of the application of the Rules. Unless otherwise specified or implied by a specific Rule that is applied, application of General Rules terminates when the last in the sequence has been applied.

— **Leveling Rules**: A specification of how the element must be supported for each of the levels of SQL.

The scope of notational symbols is the Subclause in which those symbols are defined. Within a Subclause, the symbols defined in Syntax Rules, Access Rules, or General Rules can be referenced in other rules provided that they are defined before being referenced.

3.3.3 Specification of the Information Schema

The objects of the Information Schema in this
ANSI American
ISO International
Standard are specified in terms of:

— **Function**: A short statement of the purpose of the definition.

— **Definition**: A definition, in SQL, of the object being defined.

— **Description**: A specification of the run-time value of the object, to the extent that this is not clear from the definition.

The definitions used to define the views in the Information Schema are used only to specify clearly the contents of those viewed tables. The actual objects on which these views are based are implementation-dependent.

3.3.4 Use of terms

3.3.4.1 Exceptions

The phrase “an exception condition is raised:”, followed by the name of a condition, is used in General Rules and elsewhere to indicate that the execution of a statement is unsuccessful, application of General Rules, other than those of Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard, may be terminated, diagnostic information is to be made available, and that execution of the statement is to have no effect on SQL-data or schemas. The effect on <target specification>s and SQL descriptor areas of an SQL-statement that terminates with an exception condition, unless explicitly defined by this
ANSI American
ISO International
standard, is implementation-dependent.

The phrase “a completion condition is raised:”, followed by the name of a condition, is used in General Rules and elsewhere to indicate that application of General Rules is not terminated and diagnostic information is to be made available; unless an exception condition is also raised, the execution of the statement is successful.

If more than one condition could have occurred as a result of a statement, it is implementation-dependent whether diagnostic information pertaining to more than one condition is made available. See Subclause 4.32.1, "Status parameters", for rules regarding precedence of status parameter values.

3.3.4.2 Syntactic containment

In a Format, a syntactic element <A> is said to *immediately contain* a syntactic element <B> if <B> appears on the right-hand side of the BNF production rule for <A>. A syntactic element <A> is said to *contain* or *specify* a syntactic element <C> if <A> immediately contains <C> or if <A> immediately contains a syntactic element <B> that contains <C>.

In SQL language, an instance *A1* of <A> is said to *immediately contain* an instance *B1* of <B> if <A> immediately contains <B> and the text of *B1* is part of the text of *A1*. An instance *A1* of <A> is said to *contain* or *specify* an instance *C1* of <C> if *A1* immediately contains *C1* or if *A1* immediately contains an instance *B1* of <B> that contains *C1*.

An instance *A1* of <A> is said to contain an instance *B1* of <B> *with an intervening* <C> if *A1* contains *B1* and *A1* contains an instance *C1* of <C> that contains *B1*. An instance *A1* of <A> is said to contain an instance *B1* of <B> *without an intervening* <C> if *A1* contains *B1* and *A1* does not contain an instance *C1* of <C> that contains *B1*.

An instance *A1* of <A> *simply contains* an instance *B1* of <B> if *A1* contains *B1* without an intervening instance *A2* of <A> or an intervening instance *B2* of <B>.

If <A> contains <B>, then <B> is said to be *contained in* <A> and <A> is said to be a *containing* production symbol for <B>. If <A> simply contains <B>, then <B> is said to be *simply contained in* <A> and <A> is said to be a *simply containing* production symbol for <B>.

Let *A1* be an instance of <A> and let *B1* be an instance of <B>. If <A> contains <B>, then *A1* is said to *contain B1* and *B1* is said to be *contained in A1*. If <A> simply contains <B>, then *A1* is said to *simply contain B1* and *B1* is said to be *simply contained in A1*.

An instance *A1* of <A> is the *innermost* <A> satisfying a condition *C* if *A1* satisfies *C* and *A1* does not contain an instance *A2* of <A> that satisfies *C*. An instance *A1* of <A> is the *outermost* <A> satisfying a condition *C* if *A1* satisfies *C* and *A1* is not contained in an instance *A2* of <A> that satisfies *C*.

If <A> contains a <table name> that identifies a view that is defined by a <view definition> *V*, then <A> is said to *generally contain* the <query expression> contained in *V*. If <A> contains a <routine invocation> *RI*, then <A> is said to *generally contain* all <routine>s in the set of subject <routine>s of *RI*. If <A> contains <B>, then <A> generally contains <B>. If <A> generally contains <B> and <B> generally contains <C>, then <A> generally contains <C>.

Note: The “set of subject <routine>s of a <routine invocation>” is defined in Subclause 9.1, "<routine invocation>", in Part 4 of this
ANSI American
ISO International
Standard.

An instance *A1* of <A> *directly contains* an instance *B1* of <B> if *A1* contains *B1* without an intervening <set function specification> or <subquery>.

3.3.4.3 Terms denoting rule requirements

In the Syntax Rules, the term *shall* defines conditions that are required to be true of syntactically conforming SQL language. When such conditions depend on the contents of the schema, then they are required to be true just before the actions specified by the General Rules are performed. The treatment of language that does not conform to the SQL Formats and Syntax Rules is implementation-dependent. If any condition required by Syntax Rules is not satisfied when the evaluation of Access or General Rules is attempted and the implementation is neither processing non-conforming SQL language nor processing conforming SQL language in a non-conforming manner, then an exception condition is raised: *syntax error or access rule violation*.

In the Access Rules, the term *shall* defines conditions that are required to be satisfied for the successful application of the General Rules. If any such condition is not satisfied when the General Rules are applied, then an exception condition is raised: *syntax error or access rule violation*.

In the Leveling Rules, the term *shall* defines conditions that are required to be true of SQL language for it to syntactically conform to the specified level of conformance.

3.3.4.4 Rule evaluation order

A conforming implementation is not required to perform the exact sequence of actions defined in the General Rules, but shall achieve the same effect on SQL-data and schemas as that sequence. The term *effectively* is used to emphasize actions whose effect might be achieved in other ways by an implementation.

The Syntax Rules and Access Rules for contained syntactic elements are effectively applied at the same time as the Syntax Rules and Access Rules for the containing syntactic elements. The General Rules for contained syntactic elements are effectively applied before the General Rules for the containing syntactic elements. Where the precedence of operators is determined by the Formats of this
ANSI American
ISO International
Standard or by parentheses, those operators are effectively applied in the order specified by that precedence. Where the precedence is not determined by the Formats or by parentheses, effective evaluation of expressions is *generally* performed from left to right. However, it is implementation-dependent whether expressions are *actually* evaluated left to right, particularly when operands or operators might cause conditions to be raised or if the results of the expressions can be determined without completely evaluating all parts of the expression. In general, if some syntactic element contains more than one other syntactic element, then the General Rules for contained elements that appear earlier in the production for the containing syntactic element are applied before the General Rules for contained elements that appear later.

For example, in the production:

<A> ::= <B> <C>

the Syntax Rules and Access Rules for <A>, <B>, and <C> are effectively applied simultaneously. The General Rules for <B> are applied before the General Rules for <C>, and the General Rules for <A> are applied after the General Rules for both <B> and <C>.

If the result of an expression or search condition can be determined without completely evaluating all parts of the expression or search condition, then the parts of the expression or search condition whose evaluation is not necessary are called the *inessential* parts. If the Access Rules pertaining to inessential parts are not satisfied, then the *syntax error or access rule violation* exception condition is raised regardless of whether or not the inessential parts are actually evaluated. If evaluation of

the inessential parts would cause an exception condition to be raised, then it is implementation-dependent whether or not that exception condition is raised.

3.3.4.5 Conditional rules

Conditional rules are specified with "If" or "Case" conventions. Rules specified with "Case" conventions include a list of conditional sub-rules using "If" conventions. The first such "If" sub-rule whose condition is true is the effective sub-rule of the "Case" rule. The last sub-rule of a "Case" rule may specify "Otherwise". Such a sub-rule is the effective sub-rule of the "Case" rule if no preceding "If" sub-rule in the "Case" rule has a true condition.

3.3.4.6 Syntactic substitution

In the Syntax and General Rules, the phrase "*X* is implicit" indicates that the Syntax and General Rules are to be interpreted as if the element *X* had actually been specified. Within the Syntax Rules of a given Subclause, it is known whether the element was explicitly specified or is implicit.

In the Syntax and General Rules, the phrase "the following <*X*> is implicit: *Y*" indicates that the Syntax and General Rules are to be interpreted as if a syntactic element <*X*> containing *Y* had actually been specified.

In the Syntax Rules and General Rules, the phrase "*former* is equivalent to *latter*" indicates that the Syntax Rules and General Rules are to be interpreted as if all instances of *former* in the element had been instances of *latter*.

If a BNF nonterminal is referenced in a Subclause without specifying how it is contained in a BNF production that the Subclause defines, then

Case:

— If the BNF nonterminal is itself defined in the Subclause, then the reference shall be assumed to be the occurrence of that BNF nonterminal on the left side of the defining production.

— Otherwise, the reference shall be assumed to be to a BNF production in which the particular BNF nonterminal is immediately contained.

3.3.4.7 Other terms

Some Syntax Rules define terms, such as *T1*, to denote named or unnamed tables. Such terms are used as table names or correlation names. Where such a term is used as a correlation name, it does not imply that any new correlation name is actually defined for the denoted table, nor does it affect the scopes of any actual correlation names.

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

Note: "<routine>" is defined in Part 4 of this
ANSI American
ISO International
Standard.

An <SQL procedure statement> *S1* is said to be executed as an *indirect result of executing an <SQL procedure statement>* if *S1* is a <triggered SQL statement> that is contained in some <trigger definition> and a triggering <SQL procedure statement> is executed.

ANSI Only–SQL3

An item *X is a part of* an item *Y* if and only if:

- *Y* is a row and *X* is a column of *Y*.
- *Y* is a <compound statement> and *X* is a variable specified in the <local declaration list> of *Y*.
- *Y* is a routine invocation and *X* is a parameter of *Y*.
- *Y* is an ADT instance and *X* is an attribute of *Y*.
- There exists an item *X2* such that *X* is a part of *X2* and *X2* is a part of *X*.

3.3.5 Descriptors

A descriptor is a conceptual structured collection of data that defines the attributes of an instance of an object of a specified type. The concept of descriptor is used in specifying the semantics of SQL. It is not necessary that any descriptor exist in any particular form in any database or environment.

Some SQL objects cannot exist except in the context of other SQL objects. For example, columns cannot exist except in tables. Those objects are independently described by descriptors, and the descriptors of enabling objects (e.g., tables) are said to *include* the descriptors of enabled objects (e.g., columns or table constraints). Conversely, the descriptor of an enabled object is said to *be included in* the descriptor of an enabling object.

In other cases, certain SQL objects cannot exist unless some other SQL object exists, even though there is not an inclusion relationship. For example, SQL does not permit an assertion to exist if the tables referenced by the assertion do not exist. Therefore, an assertion descriptor *is dependent on* or *depends on* one or more table descriptors (equivalently, an assertion *is dependent on* or *depends on* one or more tables). In general, a descriptor *D1* can be said to depend on, or be dependent on, some descriptor *D2*.

There are two ways of indicating dependency of one construct on another. In many cases, the descriptor of the dependent construct is said to "include the name of" the construct on which it is dependent. In this case "the name of" is to be understood as meaning 'sufficient information to identify the descriptor of'; thus an implementor might choose to use a pointer or a concatenation of <catalog name>, <schema name>, *et cetera*. Alternatively, the descriptor may be said to include text (*e.g.*, <query expression>, <search condition>). In such cases, whether the implementation includes actual text (with defaults and implications made explicit) or its own style of parse tree is irrelevant; the validity of the descriptor is clearly "dependent on" the existence of descriptors for objects that are referred to in it.

The statement that a column "is based on" a domain, is equivalent to a statement that a column "is dependent on" that domain.

An attempt to destroy a descriptor may fail if other descriptors are dependent on it, depending on how the destruction is specified. Such an attempt may also fail if the descriptor to be destroyed is included in some other descriptor. Destruction of a descriptor results in the destruction of all descriptors included in it, but has no effect on descriptors on which it is dependent.

3.3.6 Index typography

In the Index to this
ANSI American
ISO International
Standard, the following conventions are used:

— Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF nonterminal was defined;

— Index entries appearing in *italics* indicate a page where the BNF nonterminal was used in a Format; and

— Index entries appearing in roman type indicate a page where the word, phrase, or BNF nonterminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Leveling Rule, Table, or other descriptive text.

3.4 Object identifier for Database Language SQL

Function

The object identifier for Database Language SQL identifies the characteristics of an SQL-implementation to other entities in an open systems environment.

Format

```
<SQL object identifier> ::=
       <SQL provenance> <SQL variant>

<SQL provenance> ::= <arc1> <arc2> <arc3>

<arc1> ::= iso | 1 | iso <left paren> 1 <right paren>

<arc2> ::= standard | 0 | standard <left paren> 0 <right paren>

<arc3> ::= 9075

<SQL variant> ::= <SQL edition> <SQL conformance>

<SQL edition> ::= <1987> | <1989> | <1992>

<1987> ::= 0 | edition1987 <left paren> 0 <right paren>

<1989> ::= <1989 base> <1989 package>

<1989 base> ::= 1 | edition1989 <left paren> 1 <right paren>

<1989 package> ::= <integrity no> | <integrity yes>

<integrity no> ::= 0 | IntegrityNo <left paren> 0 <right paren>

<integrity yes> ::= 1 | IntegrityYes <left paren> 1 <right paren>

<1992> ::= 2 | edition1992 <left paren> 2 <right paren>
```

```
<SQL conformance> ::= <low> | <intermediate> | <high>

<low> ::= 0 | Low <left paren> 0 <right paren>

<intermediate> ::= 1 | Intermediate <left paren> 1 <right paren>

<high> ::= 2 | High <left paren> 2 <right paren>
```

****Editor's Note****
It is possible that the object identifier for SQL may have to be adjusted to account for the new structure of SQL3.

Syntax Rules

1) An <SQL conformance> of <high> shall not be specified unless the <SQL edition> is specified as <1992>.

2) The value of <SQL conformance> identifies the level at which conformance is claimed as follows:

   a) If <SQL edition> specifies <1992>, then

      Case:

      i) <low>, then Entry SQL level.

      ii) <intermediate>, then Intermediate SQL level.

      iii) <high>, then Full SQL level.

   b) Otherwise:

      i) <low>, then level 1.

      ii) <intermediate>, then level 2.

3) A specification of <1989 package> as <integrity no> implies that the integrity enhancement feature is not implemented. A specification of <1989 package> as <integrity yes> implies that the integrity enhancement feature is implemented.

4 Concepts

• 1 Subclause moved to Part 1

4.1 Data types

A data type is a set of representable values. The logical representation of a value is a <literal>. The physical representation of a value is implementation-dependent.

SQL supports two sorts of data types: *predefined data types* and *abstract data types*. Predefined data types are sometimes called *built-in data types*. Abstract data types can be defined by a standard, by an implementation, or by an application. Abstract data types have no corresponding <literal>s and thus no logical representation. Abstract data types are summarized in Subclause 11.47, "<abstract data type definition>".

New primitive data types or abstract data types can be defined by means of distinct types. A distinct type is a new type that shares the same representation as the type from which it is derived, the so-called *source type*.

SQL also supports the collection data types of set, multiset and list. A collection is a multiset of values or objects called elements. The elements may be primitive data type elements, abstract data type elements or collections. A set is an unordered collection of distinct elements, with no duplicates. A multiset is similar to a set except that duplicates are permitted. A list is similar to a multiset except that the elements are ordered.

SQL further supports SQL object identifier data types associated with the OID attribute of each object ADT. These data types cannot be used except implicitly as specified in this
ANSI American
ISO International
standard.

• 2 paragraphs deleted

Values are either null values or non-null values.

A null value is an implementation-dependent special value that is distinct from all non-null values of the associated data type. There is effectively only one null value and that value is a member of every SQL data type. There is no <literal> for a null value although the keyword NULL is used in some places to indicate that a null value is desired.

SQL defines distinct data types named by the following <key word>s: CHARACTER, CHARACTER VARYING, CHARACTER LARGE OBJECT, BINARY LARGE OBJECT, BIT, BIT VARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, ENUMERATED, FLOAT, REAL, DOUBLE PRECISION, BOOLEAN, DATE, TIME, TIMESTAMP, and INTERVAL.

For reference purposes, the data types CHARACTER, CHARACTER VARYING, and CHARACTER LARGE OBJECT are collectively referred to as *character string types*. The data types BIT and BIT VARYING are collectively referred to as *bit string types*. The data type BINARY LARGE OBJECT is referred to as the *binary string type* and the values of binary string types are referred to as *binary strings*. The data types CHARACTER LARGE OBJECT and BINARY LARGE OBJECT are collectively referred to as *large object string types* and the values of large object string types are referred to as *large object strings*. Character string types, bit string types, and binary string types are collectively referred to as *string types* and values of string types are referred to as *strings*.

The data types NUMERIC, DECIMAL, INTEGER and SMALLINT are collectively referred to as *exact numeric types*. The data types FLOAT, REAL, and DOUBLE PRECISION are collectively referred to as *approximate numeric types*. Exact numeric types and approximate numeric types are collectively referred to as *numeric types*. Values of numeric types are referred to as *numbers*. The data types DATE, TIME, and TIMESTAMP are collectively referred to as *datetime types*. Values of datetime types are referred to as *datetimes*. The data type INTERVAL is referred to as an *interval type*. Values of interval types are called *intervals*.

Each data type has an associated data type descriptor; the contents of a data type descriptor are determined by the specific data type that it describes. A data type descriptor includes an identification of the data type and all information needed to characterize an instance of that data type.

Subclause 6.1, "<data type>", describes the semantic properties of each data type.

Each host language has its own data types, which are separate and distinct from SQL data types, even though similar names may be used to describe the data types. Mappings of SQL data types to data types in host languages are described in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard, and Subclause 14.1, "<embedded SQL host program>", in Part 5 of this
ANSI American
ISO International
Standard. Not every SQL data type has a corresponding data type in every host language.

4.2 Character strings

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

— the name of the specific character string data 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 string data type;

— the catalog name, schema name, and character set name of the character set of the character string data type; and

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

Character sets fall into three categories: those defined by national or international standards, those provided by implementations, and those defined by applications. All character sets, however defined, always contain the <space> character. Character sets defined by applications can be defined to “reside” in any schema chosen by the application. Character sets defined by standards or by implementations reside in the Information Schema (named INFORMATION_SCHEMA) in each catalog, as do collations defined by standards and collations, translations, and form-of-use conversions defined by implementations.

The <implementation-defined character repertoire name> SQL_TEXT specifies the name of a character repertoire and implied form-of-use that can represent every character that is in <SQL language character> and all other characters that are in character sets supported by the implementation.

4.2.1 Character strings and collating sequences

A character string is a sequence of characters chosen from the same character repertoire. The character repertoire from which the characters of a particular string are chosen may be specified explicitly or implicitly. A character string has a length, which is the number of characters in the sequence. The length is 0 or a positive integer.

All character strings of a given character repertoire are comparable.

A collating sequence, also known as a collation, is a set of rules determining comparison of character strings in a particular character repertoire. There is a default collating sequence for each character repertoire, but additional collating sequences can be defined for any character repertoire.

Note: A column may be defined as having a default collating sequence. This default collating sequence for the column may be different from the default collating sequence for its character repertoire, *e.g.*, if the <collate clause> is specified in the column reference. It will be clear from context when the term "default collating sequence" is used whether it is meant for a column or for a character repertoire.

Given a collating sequence, two character strings are identical if and only if they are equal in accordance with the comparison rules specified in Subclause 8.2, "<comparison predicate>". The collating sequence used for a particular comparison is determined as in Subclause 4.2.3, "Rules determining collating sequence usage".

The <key word>s NATIONAL CHARACTER are used to specify a character string data type with a particular implementation-defined character repertoire. Special syntax (N"string") is provided for representing literals in that character repertoire.

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

— the name of the character set or character repertoire,

— if the character set is a character repertoire, then the name of the form-of-use,

— an indication of what characters are in the character set, and

— whether or not the character set uses the DEFAULT collation for its character repertoire, and

— if the character set does not utilize the DEFAULT collation for its character repertoire, then the <translation name> conained in the character set's <translation collation>, if any, the <collation name> contained in the character set's <collate clause> or <limited collation definition>, if any, and whether or not DESC was specified in the reference to the collation.

For every character set, there is at least one collation. A collation is described by a collation descriptor. A collation descriptor includes:

— the name of the collation,

— the name of the character repertoire on which the collation operates,

— whether the collation has the NO PAD or the PAD SPACE attribute, and

— whether or not this collation utilizes the DEFAULT collation for its character repertoire,

— if the collation does not utilize the DEFAULT collation for its character repertoire, then the <translation name> contained in the collation's <translation definition>, if any, the <collation name> contained in the collation's <collation source>, if any, and whether or not DESC was specified in the definition of the collation.

4.2.2 Operations involving character strings

4.2.2.1 Operators that operate on character strings and return character strings

<concatenation operator> is an operator, ||, that returns the character string made by joining its character string operands in the order given.

<character substring function> is a triadic function, SUBSTRING, that returns a string extracted from a given string according to a given numeric starting position and a given numeric length. Truncation occurs when the implied starting and ending positions are not both within the given string.

<character overlay function> is a function, OVERLAY, that modifies a string argument by replacing a given substring of the string, which is specified by a given numeric starting position and a given numeric length, with another string (called the replacement string). When the length of the substring is zero, nothing is removed from the original string and the string returned by the function is the result of inserting the replacement string into the original string at the starting position.

<fold> is a pair of functions for converting all the lower case characters in a given string to upper case (UPPER) or all the upper case ones to lower case (LOWER), useful only in connection with strings that may contain <simple Latin letter>s.

<form-of-use conversion> is a function that invokes an installation-supplied form-of-use conversion to return a character string *S2* derived from a given character string *S1*. It is intended, though not enforced by this
ANSI American
ISO International
Standard, that *S2* be exactly the same sequence of characters as *S1*, but encoded according some different form-of-use. A typical use might be to convert a character string from two-octet UCS to one-octet Latin1 or *vice versa*.

<trim function> is a function that returns its first string argument with leading and/or trailing pad characters removed. The second argument indicates whether leading, or trailing, or both leading and trailing pad characters should be removed. The third argument specifies the pad character that is to be removed.

<character translation> is a function for changing each character of a given string according to some many-to-one or one-to-one mapping between two not necessarily distinct character sets. The mapping, rather than being specified as part of the function, is some external function identified by a <translation name>.

For any pair of character sets, there are zero or more translations that may be invoked by a <character translation>. A translation is described by a translation descriptor. A translation descriptor includes:

— the name of the translation,

— the name of the character set from which it translates,

— the name of the character set to which it translates, and

— an indication of how the translation is performed.

4.2.2.2 Other operators involving character strings

<length expression> returns the length of a given character string, as an exact numeric value, in characters, octets, or bits according to the choice of function.

<position expression> determines the first position, if any, at which one string, *S1*, occurs within another, *S2*. If *S1* is of length zero, then it occurs at position 1 for any value of *S2*. If *S1* does not occur in *S2*, then zero is returned. The data type of the result of a <position expression> is exact numeric.

<like predicate> uses the triadic operator LIKE (or the inverse, NOT LIKE), operating on three character strings and returning a Boolean. LIKE determines whether or not a character string "matches" a given "pattern" (also a character string). The characters "%" (percent) and "_" (underscore) have special meaning when they occur in the pattern. The optional third argument is a character string containing exactly one character, known as the "escape character", for use when a percent or underscore is required in the pattern without its special meaning.

4.2.2.3 Operations involving large object character strings

Large object strings can not participate in most comparison operations. Large object strings can, however, participate in the following comparison operations:

— <like predicate>

— <position expression>

— <comparison predicate> with an <equals operator> or <not equals operator>

— <quantified comparison predicate> with the <equals operator> or <not equals operator>

As a result of these restrictions, large object strings and large object string columns cannot be referenced in (among other places):

— predicates other than those listed above and the <exists predicate>

— <general set function>

— <group by clause>

— <order by clause>

— <unique constraint definition>,

— <referential constraint definition>,

— <select list> of a <query specification> that has a <set quantifier> of DISTINCT,

— UNION, INTERSECT, and EXCEPT, and

— columns used for matching when forming a <joined table>.

All the operations described within Subclause 4.2.2.1, "Operators that operate on character strings and return character strings", and Subclause 4.2.2.2, "Other operators involving character strings", are supported for large object character strings.

4.2.3 Rules determining collating sequence usage

The rules determining collating sequence usage for character strings are based on the following:

— Expressions where no columns are involved (*e.g.*, literals, host variables) are by default compared using the default collating sequence for their character repertoire.

Note: The default collating sequence for a character repertoire is defined in Subclause 10.5, "<character set specification>", and Subclause 11.37, "<character set definition>".

— When one or more columns are involved (*e.g.*, comparing two columns, or comparing a column to a literal), then provided that all columns involved have the same default collating sequence and there is no explicit specification of a collating sequence, that default collating sequence is used.

— When columns are involved having different default collating sequences, explicit specification of the collating sequence in the expression is required via the <collate clause>.

— Any explicit specification of collating sequence in an expression overrides any default collating sequence.

To formalize this, <character value expression>s effectively have a coercibility attribute. This attribute has the values *Coercible*, *Implicit*, *No collating sequence*, and *Explicit*. <character value expression>s with the *Coercible*, *Implicit*, or *Explicit* attributes have a collating sequence.

A <character value expression> consisting of a column reference has the coercibility attribute *Implicit*, with collating sequence as defined when the column was created. A <character value expression> consisting of a value other than a column (*e.g.*, a host variable or a literal) has the coercibility attribute *Coercible*, with the default collation for its character repertoire. A <character value expression> simply containing a <collate clause> has the coercibility attribute *Explicit*, with the collating sequence specified in the <collate clause>.

Note: When the coercibility attribute is *Coercible*, the collating sequence is uniquely determined as specified in Subclause 8.2, "<comparison predicate>".

The tables below define how the collating sequence and the coercibility attribute is determined for the result of any monadic or dyadic operation. Table 1, "Collating coercibility rules for monadic operators", shows the collating sequence and coercibility rules for monadic operators, and Table 2, "Collating coercibility rules for dyadic operators", shows the collating sequence and coercibility rules for dyadic operators. Table 3, "Collating sequence usage for comparisons", shows how the collating sequence is determined for a particular comparison.

Table 1—Collating coercibility rules for monadic operators

Operand Coercibility and Collating Sequence		Result Coercibility and Collating Sequence	
Coercibility	**Collating Sequence**	**Coercibility**	**Collating Sequence**
Coercible	default	Coercible	default
Implicit	*X*	Implicit	*X*
Explicit	*X*	Explicit	*X*
No collating sequence		No collating sequence	

Table 2—Collating coercibility rules for dyadic operators

Operand 1 Coercibility and Collating Sequence		Operand 2 Coercibility and Collating Sequence		Result Coercibility and Collating Sequence	
Coercibility	**Collating Sequence**	**Coercibility**	**Collating Sequence**	**Coercibility**	**Collating Sequence**
Coercible	default	Coercible	default	Coercible	default
Coercible	default	Implicit	*Y*	Implicit	*Y*
Coercible	default	No collating sequence		No collating sequence	
Coercible	default	Explicit	*Y*	Explicit	*Y*
Implicit	*X*	Coercible	default	Implicit	*X*
Implicit	*X*	Implicit	*X*	Implicit	*X*
Implicit	*X*	Implicit	$Y \neq X$	No collating sequence	
Implicit	*X*	No collating sequence		No collating sequence	
Implicit	*X*	Explicit	*Y*	Explicit	*Y*
No collating sequence		Any, except Explicit	Any	No collating sequence	
No collating sequence		Explicit	*X*	Explicit	*X*
Explicit	*X*	Coercible	default	Explicit	*X*
Explicit	*X*	Implicit	*Y*	Explicit	*X*
Explicit	*X*	No collating sequence		Explicit	*X*
Explicit	*X*	Explicit	*X*	Explicit	*X*
Explicit	*X*	Explicit	$Y \neq X$	Not permitted: *invalid syntax*	

Table 3—Collating sequence usage for comparisons

Comparand 1 Coercibility and Collating Sequence		Comparand 2 Coercibility and Collating Sequence		
Coercibility	**Collating Sequence**	**Coercibility**	**Collating Sequence**	**Collating Sequence Used For The Comparison**
Coercible	default	Coercible	default	default
Coercible	default	Implicit	*Y*	*Y*
Coercible	default	No collating sequence		Not permitted: *invalid syntax*
Coercible	default	Explicit	*Y*	*Y*
Implicit	*X*	Coercible	default	*X*
Implicit	*X*	Implicit	*X*	*X*
Implicit	*X*	Implicit	$Y \neq X$	Not permitted: *invalid syntax*
Implicit	*X*	No collating sequence		Not permitted: *invalid syntax*
Implicit	*X*	Explicit	*Y*	*Y*
No collating sequence		Any except Explicit	Any	Not permitted: *invalid syntax*
No collating sequence		Explicit	*X*	*X*
Explicit	*X*	Coercible	default	*X*
Explicit	*X*	Implicit	*Y*	*X*
Explicit	*X*	No collating sequence		*X*
Explicit	*X*	Explicit	*X*	*X*
Explicit	*X*	Explicit	$Y \neq X$	Not permitted: *invalid syntax*

For *n*-adic operations (*e.g.*, <case expression>) with operands X_1, X_2, . . . , X_n, the collating sequence is effectively determined by considering X_1 and X_2, then combining this result with X_3, and so on.

4.3 Binary strings

A binary string is a sequence of octets that does not have either a character set or collation associated with it.

A binary data type is described by a binary data type descriptor. A binary data type descriptor contains:

— the name of the data type (BINARY LARGE OBJECT); and

— the maximum length of the binary string data type (in octets).

4.3.1 Binary string comparison

All binary strings are mutually comparable. A binary string is identical to another binary string if and only if it is equal to that binary string in accordance with the comparison rules specified in Subclause 8.2, "<comparison predicate>".

4.3.2 Operations involving binary strings

4.3.2.1 Operators that operate on binary strings and return binary strings

<blob concatenation> is an operator, ||, that returns a binary string by joining its binary string operands in the order given.

<blob substring function> is a triadic function identical in syntax and semantics to <character substring function> except that the returned value is a binary string.

<blob overlay function> is a function identical in syntax and semantics to <character overlay function> except that the first argument, second argument, and returned value are all binary strings.

<trim function> is identical in syntax and sematics as when operating on character strings except that the returned value is a binary string.

4.3.2.2 Other operators involving binary strings

<length expression> returns the length of a given binary string, as an exact numeric value, in characters, octets, or bits according to the choice of function.

<position expression> is identical in syntax and semantics as when operating on character strings except that the operands are binary strings.

<like predicate> is identical in syntax and semantics as when operating on character strings except that the operands are binary strings.

4.4 Bit strings

A bit string is a sequence of bits, each having the value of 0 or 1. A bit string has a length, which is the number of bits in the string. The length is 0 or a positive integer.

A bit string data type is described by a bit string data type descriptor. A bit string data type descriptor contains:

— the name of the specific bit string data type (BIT or BIT VARYING); and

— the length of the bit string data type (in bits).

4.4.1 Bit string comparison and assignment

All bit strings are mutually comparable. A bit string is identical to another bit string if and only if it is equal to that bit string in accordance with the comparison rules specified in Subclause 8.2, "<comparison predicate>".

• 1 paragraph deleted.

4.4.2 Operations involving bit strings

4.4.2.1 Operators that operate on bit strings and return bit strings

<bit concatenation> is an operator, ||, that returns the bit string made by concatenating the two bit string operands in the order given.

<bit substring function> is a triadic function identical in syntax and semantics to <character substring function> except that the first argument and the returned value are both bit strings.

4.4.2.2 Other operators involving bit strings

<length expression> returns the length (as an integer number of octets or bits according to the choice of function) of a given bit string.

<position expression> determines the first position, if any, at which one string, *S1*, occurs within another, *S2*. If *S1* is of length zero, then it occurs at position 1 for any value of *S2*. If *S1* does not occur in *S2*, then zero is returned.

4.5 Large object locators

A *large object locator*, or *locator*, is a value generated when a large object string is assigned to an <embedded variable name> of a large object locator type. A locator uniquely identifies the string value that was assigned to it.

When an <embedded variable name> of a large object locator type is used in a <value specification>, it is equivalent to the value of the large object string that was assigned to it.

There are two different types of locators:

— Binary large object locator type, whose values are used to identify binary strings.

— Character large object locator type, whose values are used to identify values of character large object type.

In a host variable, a locator is materialized as a four octet value that can be represented by a four byte integer.

A locator type can never be specified as the <data type> of columns, and as a result, the database system does not treat large object locators as a persistent type.

A locator identifies the large object value that was assigned to it within the SQL-transaction in which the locator was established. At the end of an SQL-transaction all locators established within the current SQL-transaction are destroyed.

Locators are part of the context of an SQL-session.

4.6 Numbers

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

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

— the name of the specific numeric data type (NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, or DOUBLE PRECISION);

— the precision of the numeric data type;

— the scale of the numeric data type, if it is an exact numeric data type; and

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

A value described by a numeric data type descriptor is always signed.

4.6.1 Characteristics of numbers

An exact numeric value has a precision and a scale. The precision is a positive integer that determines the number of significant digits in a particular radix (binary or decimal). The scale is a non-negative integer. A scale of 0 indicates that the number is an integer. For a scale of S, the exact numeric value is the integer value of the significant digits multiplied by 10^{-S}.

An approximate numeric value consists of a mantissa and an exponent. The mantissa is a signed numeric value, and the exponent is a signed integer that specifies the magnitude of the mantissa. An approximate numeric value has a precision. The precision is a positive integer that specifies the number of significant binary digits in the mantissa. The value of an approximate numeric value is the mantissa multiplied by 10^x, where x is the exponent.

Whenever an exact or approximate numeric value is assigned to a data item or parameter representing an exact numeric value, an approximation of its value that preserves leading significant digits after rounding or truncating is represented in the data type of the target. The value is converted to have the precision and scale of the target. The choice of whether to truncate or round is implementation-defined.

An approximation obtained by truncation of a numeric value N for an <exact numeric type> T is a value V in T such that N is not closer to zero than V and there is no value in T between V and N.

An approximation obtained by rounding of a numeric value N for an <exact numeric type> T is a value V in T such that the absolute value of the difference between N and the numeric value of V is not greater than half the absolute value of the difference between two successive numeric values in T. If there is more than one such value V, then it is implementation-defined which one is taken.

All numeric values between the smallest and the largest value, inclusive, in a given exact numeric type have an approximation obtained by rounding or truncation for that type; it is implementation-defined which other numeric values have such approximations.

An approximation obtained by truncation or rounding of a numeric value N for an <approximate numeric type> T is a value V in T such that there is no numeric value in T and distinct from that of V that lies between the numeric value of V and N, inclusive.

If there is more than one such value V then it is implementation-defined which one is taken. It is implementation-defined which numeric values have approximations obtained by rounding or truncation for a given approximate numeric type.

Whenever an exact or approximate numeric value is assigned to a data item or parameter representing an approximate numeric value, an approximation of its value is represented in the data type of the target. The value is converted to have the precision of the target.

Operations on numbers are performed according to the normal rules of arithmetic, within implementation-defined limits, except as provided for in Subclause 6.17, "<numeric value expression>".

4.6.2 Operations involving numbers

As well as the usual arithmetic operators, plus, minus, times, divide, unary plus, and unary minus, there are the following functions that return numbers:

— <position expression> (see Subclause 4.2.2, "Operations involving character strings", and Subclause 4.4.2, "Operations involving bit strings") takes two strings as arguments and returns an integer;

— <length expression> (see Subclause 4.2.2, "Operations involving character strings", and Subclause 4.4.2, "Operations involving bit strings") operates on a string argument and returns an integer;

— <extract expression> (see Subclause 4.9.3, "Operations involving datetimes and intervals") operates on a datetime or interval argument and returns an integer.

4.7 Enumerated types

An enumerated type is a list of distinct identifiers that represents an ordered set of values. All values for a given enumerated type are comparable. Values of two different enumerated types are not comparable.

An enumerated data type is described by an enumerated data type descriptor. An enumerated data type descriptor contains:

— the name of the enumerated data type (ENUMERATED);

— the number of enumeration names that participate in the enumerated type; and

— a list of the enumeration names that participate in the enumerated type.

4.8 Boolean types

The data type boolean comprises the values *true* and *false* .

The boolean data type is described by the boolean data type descriptor. The boolean data type descriptor contains:

— the name of the boolean data type (BOOLEAN).

4.8.1 Comparison and assignment of booleans

Value of boolean data type are comparable, and either boolean value can be assigned to a boolean variable

4.8.2 Operations involving booleans

4.8.2.1 Operations on booleans that return booleans

The monadic boolean operator NOT and the dynadic boolean operators AND and OR take boolean operands and produce a boolean result.

4.8.2.2 Other operators involving booleans

The data type of the result of every <predicate> is boolean.

4.9 Datetimes and intervals

A datetime data type is described by a datetime data type descriptor. An interval data type is described by an interval data type descriptor.

A datetime data type descriptor contains:

— the name of the specific datetime data type (DATE, TIME, TIMESTAMP, TIME WITH TIME ZONE, or TIMESTAMP WITH TIME ZONE); and

— the value of the <time fractional seconds precision>, if it is a TIME, TIMESTAMP, TIME WITH TIME ZONE, or TIMESTAMP WITH TIME ZONE type.

An interval data type descriptor contains:

— the name of the interval data type (INTERVAL);

— an indication of whether the interval data type is a year-month interval or a day-time interval; and

— the <interval qualifier> that describes the precision of the interval data type.

A value described by an interval data type descriptor is always signed.

Every datetime or interval data type has an implied *length in positions*. Let *D* denote a value in some datetime or interval data type *DT*. The length in positions of *DT* is constant for all *D*. The length in positions is the number of characters from the character set SQL_TEXT that it would take to represent any value in a given datetime or interval data type.

4.9.1 Datetimes

Table 4, "Fields in datetime items", specifies the fields that can make up a datetime value; a datetime value is made up of a subset of those fields. Not all of the fields shown are required to be in the subset, but every field that appears in the table between the first included primary field and the last included primary field shall also be included. If either timezone field is in the subset, then both of them shall be included.

Table 4—Fields in datetime items

Keyword	Meaning
Primary datetime fields	
YEAR	Year
MONTH	Month within year
DAY	Day within month
HOUR	Hour within day
MINUTE	Minute within hour
SECOND	Second and possibly fraction of a second within minute
Timezone datetime fields	
TIMEZONE_HOUR	Hour value of time zone displacement
TIMEZONE_MINUTE	Minute value of time zone displacement

There is an ordering of the significance of <datetime field>s. This is, from most significant to least significant: YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

The <datetime field>s other than SECOND contain non-negative integer values, constrained by the natural rules for dates using the Gregorian calendar. SECOND, however, can be defined to have a <time fractional seconds precision> that indicates the number of decimal digits maintained following the decimal point in the seconds value, a non-negative exact numeric value.

There are three classes of datetime data types defined within this
ANSI American
ISO International
Standard:

— DATE — contains the <datetime field>s YEAR, MONTH, and DAY;

— TIME — contains the <datetime field>s HOUR, MINUTE, and SECOND; and

— TIMESTAMP — contains the <datetime field>s YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND.

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

Datetimes only have absolute meaning in the context of additional information. Time zones are political divisions of the earth's surface that allow the convention that time is measured the same at all locations within the time zone, regardless of the precise value of "sun time" at specific locations. Political entities often change the "local time" within a time zone for certain periods of the year, e.g., in the summer. However, different political entities within the same time zone are not necessarily synchronized in their local time changes. When a datetime is specified (in SQL-data or elsewhere) it has an implied or explicit time zone specifier associated with it. Unless that time zone specifier, and its meaning, is known, the meaning of the datetime value is ambiguous.

Therefore, datetime data types that contain time fields (TIME and TIMESTAMP) are maintained in Universal Coordinated Time (UTC), with an explicit or implied time zone part.

The time zone part is an interval specifying the difference between UTC and the actual date and time in the time zone represented by the time or timestamp data item. The time zone displacement is defined as

```
INTERVAL HOUR TO MINUTE
```

A TIME or TIMESTAMP that does not specify WITH TIME ZONE has an implicit time zone equal to the local time zone for the SQL-session. The value of time represented in the data changes along with the local time zone for the SQL-session. However, the *meaning* of the time does not change because it is effectively maintained in UTC.

Note: On occasion, UTC is adjusted by the omission of a second or the insertion of a "leap second" in order to maintain synchronization with sidereal time. This implies that sometimes, but very rarely, a particular minute will contain exactly 59, 61, or 62 seconds.

4.9.2 Intervals

There are two classes of intervals. One class, called *year-month* intervals, has an express or implied datetime precision that includes no fields other than YEAR and MONTH, though not both are required. The other class, called *day-time* intervals, has an express or implied interval precision that can include any fields other than YEAR or MONTH.

Table 5, "Fields in year-month INTERVAL items", specifies the fields that make up a year-month interval. A year-month interval is made up of a contiguous subset of those fields.

Table 5—Fields in year-month INTERVAL items

Keyword	Meaning
YEAR	Years
MONTH	Months

Table 6, "Fields in day-time INTERVAL items", specifies the fields that make up a day-time interval. A day-time interval is made up of a contiguous subset of those fields.

Table 6—Fields in day-time INTERVAL items

Keyword	Meaning
DAY	Days
HOUR	Hours
MINUTE	Minutes
SECOND	Seconds and possibly fractions of a second

The actual subset of fields that comprise an item of either type of interval is defined by an <interval qualifier> and this subset is known as the precision of the item.

Within an item of type interval, the first field is constrained only by the <interval leading field precision> of the associated <interval qualifier>. Table 7, "Valid values for fields in INTERVAL items", specifies the constraints on subsequent field values.

Table 7—Valid values for fields in INTERVAL items

Keyword	Valid values of INTERVAL fields
YEAR	Unconstrained except by <interval leading field precision>
MONTH	Months (within years) (0-11)
DAY	Unconstrained except by <interval leading field precision>
HOUR	Hours (within days) (0-23)
MINUTE	Minutes (within hours) (0-59)
SECOND	Seconds (within minutes) (0-59.999...)

Values in interval fields other than SECOND are integers. SECOND, however, can be defined to have an <interval fractional seconds precision> that indicates the number of decimal digits maintained following the decimal point in the seconds value.

Fields comprising an item of type interval are also constrained by the definition of the Gregorian calendar.

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

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

4.9.3 Operations involving datetimes and intervals

Table 8, "Valid operators involving datetimes and intervals", specifies the data types of the results of arithmetic expressions involving datetime and interval operands.

Table 8—Valid operators involving datetimes and intervals

Operand 1	Operator	Operand 2	Result Type
Datetime	−	Datetime	Interval
Datetime	+ or −	Interval	Datetime
Interval	+	Datetime	Datetime
Interval	+ or −	Interval	Interval
Interval	* or /	Numeric	Interval
Numeric	*	Interval	Interval

Arithmetic operations involving items of type datetime or interval obey the natural rules associated with dates and times and yield valid datetime or interval results according to the Gregorian calendar.

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

Operations involving a datetime and an interval preserve the time zone of the datetime operand. If the datetime operand does not include a time zone part, then the local time zone is effectively used.

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

<extract expression> operates on a datetime or interval and returns an exact numeric value representing the value of one component of the datetime or interval.

4.10 Object identifier

An object identifier *OID* is a value generated when an object is created, to give that object an immutable identity. It is unique in the known universe of objects that are instances of abstract data types, and is conceptually separate from the value, or state, of the instance.

ISO Only—caused by ANSI changes not yet considered by ISO

The object identifier type is described by an object identifier type descriptor. An object identifier type descriptor contains:

— an indication that this is an object identifier type; and

— the name of the abstract data type within which the object identifier type is used.

The object identifier type is only used to define the OID pseudo-column implicitly defined in object ADTs within an ADT definition.

An OID literal exists for an object identifier type only if the associated abstract data type was defined WITH OID VISIBLE. The OID value is materialized as a character string with an implementation-defined length and character set SQL_TEXT.

- 1 Subclause deleted; contents moved

4.11 Abstract data types

An abstract data type (ADT) is a data type whose characteristics are specified by an abstract data type descriptor. An abstract data type descriptor specifies a set of declarations of the attributes that represent the value of the abstract data type, the operations that define the equality and ordering relationships of the abstract data type, and the operations and derived attributes that implement the behavior of the abstract data type. The operations and derived attributes are implemented by

<routine>s (see Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard and Subclause 9.1, "<routine invocation>"), in Part 4 of this
ANSI American
ISO International
Standard.

A value of an abstract data type ADT_i is comparable to another value of data type ADT_j if and only if ADT_i and ADT_j are in the same subtype family.

A value of an abstract data type ADT_i is assignable to an item of abstract data type ADT_j if and only if ADT_i is a subtype of ADT_j.

Note: "Subtype" is defined in Subclause 4.11.5, "Subtypes and supertypes for ADTs".

An abstract data type is either a component of a <module> or an object in a schema.

An abstract data type is described by an abstract data type descriptor. An abstract data type descriptor contains:

— The name of the abstract data type;

— The ordering specification for the abstract data type:

- If RELATIVE is specified, then the relative ordering function;
- If HASH is specified, then the hash ordering function;
- If EQUALS is specified, then the name of the equals function;
- If LESS THAN is specified, then the name of the less-than function;

— If the abstract data type has one or more cast operations, then the names of each of those cast operations;

— If the abstract data type is a direct subtype of one or more other abstract data types, then the names of those abstract data types;

— The attribute descriptor of every attribute of the abstract data type;

— The descriptor of each table constraint specified for the type;

— The degree of the abstract data type (the number of column descriptors);

— An indication of whether the abstract data type is VALUE or OBJECT, and if it is OBJECT, whether it is WITH OID VISIBLE or WITH OID NOT VISIBLE;

— The function descriptor of every operation that has the abstract data type as a parameter or result;

— The value of the <default option>, if any, of the abstract data type.

— Other information To Be Supplied.

4.11.1 Distinct types

A *distinct type* is a type whose internat representation is derived from an existing type. The type from which the distinct type is derived is denoted the *source type* of that distinct type.

A distinct type is described by a distinct type descriptor. Every distinct type descriptor includes:

— the name of the distinct type;

— the name of the source type of the distinct type; and

— an indication of whether the source type is a predefined data type or an abstract data type.

In addition to the components of every distinct type descriptor, the descriptor of a distinct type whose source type is an abstract data type includes the same components as the descriptor of its source type.

4.11.2 Encapsulation

Each component of an abstract data type has its encapsulation level specified as either public, private, or protected. Public components form the interface of the ADT and are visible to all authorized users of the ADT. Private components are totally encapsulated and are visible only within the definition of the ADT that contains them. Protected components are partially encapsulated, being visible both within their own ADT and within the definitions of all subtypes of that ADT.

Encapsulation is a property of the structure of the ADT definitions, independently of questions of which users are privileged to create or use ADTs (see Subclause 4.37, "Privileges and roles"). Once it is determined what components are public and are thus in principle accessible outside an ADT definition, the privilege mechanisms are applied to control whether a given user is authorized to use a given public component.

4.11.3 Observers and mutators

When an ADT is defined, an observer and a mutator function is implicitly defined for every attribute of the ADT. The observer function has the same name as the attribute, takes a single argument whose type is the ADT being defined, and returns the value of the attribute.

The mutator function has the same name as the attribute and two parameters. The type of the first parameter is the ADT being defined and the type of the second parameter is the data type of the attribute. The function modifies the value of the attribute with the value of the second argument and returns this value.

The encapsulation level of the observer and mutator function is defined by the encapsulation level of the attribute with which they are associated.

4.11.4 Constructors

When an ADT is defined, a constructor function is automatically defined by the system to create new instances of the type. The constructor function has the same name as the type and takes zero arguments. It returns a new instance of the
ISO type, whose OID field (if any) is set, and
type whose attributes are uninitialized. The constructor function is PUBLIC.

4.11.5 Subtypes and supertypes for ADTs

An abstract data type T_a is a subtype of another abstract data type T_b if the abstract data type descriptor for T_a indicates that it is a subtype of T_b. If abstract data type T_a is declared to be a subtype of another abstract data type T_b by an <abstract data type definition>, then T_a is called a *direct subtype* of T_b.

Type T_a is a subtype of type T_b if and only if either:

— T_a and T_b are the same named type,

— T_a is a direct subtype of T_b, or

— there is a type T_c such that T_a is a direct subtype of T_c and T_c is a subtype of T_b.

A type T is considered to be one of its own subtypes. Subtypes of T other than T itself are called its *proper subtypes*.

Type T_b is called a *supertype* of T_a if T_a is a subtype of T_b. If T_a is a direct subtype of T_b, then T_b is called a *direct supertype* of T_a. A type that is not a subtype of any other type is called a *maximal supertype*. A supertype shall not have itself as a proper subtype.

Let T_a be a maximal supertype and let T be a subtype of T_a. The set of all subtypes of T_a (which includes T_a itself) is called a *subtype family* of T or (equivalently) of T_a.

- 1 sentence deleted

A *leaf type* is a type that does not have any subtypes.

Any instance of a subtype is an instance of all of its supertypes. An instance is associated with exactly one *most specific type*, which is defined by the property that every other type associated with the instance is a supertype of the most specific type. The most specific type of an instance need not correspond to a leaf type. For example, a type structure might consist of a type PERSON that has STUDENT and EMPLOYEE as its two subtypes, while STUDENT has two direct subtypes UG_STUDENT and PG_STUDENT. An instance in this type structure might be created with a most specific type of STUDENT, even though it does not correspond to a leaf type.

If T_a is a subtype of T_b, then an instance of T_a can be used wherever an instance of T_b is expected. In particular, an instance of T_a can be assigned to a variable of type T_b, passed as an argument for an input parameter declared as type T_b, and returned from a function whose result type is declared to be T_b.

A type T is said to be the most specific unique type of a set of types S in a subtype family if T is a supertype of every type in S and a subtype of every type that is a supertype of every type in S.

Subclause 4.11.5, "Subtypes and supertypes for ADTs", describes the capabilities of subtypes.

Users must have the UNDER privilege on a type before they can use the type in a subtype definition. A type can have more than one proper subtype. Similarly, a type can have more than one proper supertype. Thus, a subtype is a specialized type of one or more supertypes and a supertype is a generalized type of one or more subtypes.

A subtype definition has lexical access to the representation of all of its direct supertypes (but only within the ADT definition that defines the subtype of that supertype), but it has no lexical access to the representation of its sibling types. Effectively, components of all direct supertype representations are copied to the subtype's representation with same name and data type. To avoid name clashes, a subtype can rename selected components of the representation inherited from its direct supertypes.

A subtype can define actor and destructor functions like any other ADT. A subtype can also define operations which have the same name as operations defined on other types, including its supertypes.

4.11.6 Type templates

A type template is used to define a family of abstract data types. Each type template definition specifies a set of formal parameters and a body. The parameters specify values or data types that must be provided when the type template is used to designate a specific generated type. The type template body is identical to that of an abstract data type definition, except that it may contain the formal parameters of the type template. A type template is described by a type template descriptor which contains:

— the name of the type template,

— the name, <template parameter type>, and position of each of the type template's parameters, and

— the body of the type template.

A generated type is an abstract data type resulting from a <generated type reference>. A <generated type reference> specifies the type template name and a set of actual parameters. Each actual parameter must be a <literal>, a <data type>, or some other formal parameter whose actual parameter (directly or indirectly) is in turn a <literal> or <data type>.

A <generated type reference> may be specified wherever a <data type> is allowed; thus nesting of <generated type reference>s is also allowed.

Type template names may be overloaded; that is, more than one type template with the same name may be defined. To define such type templates, their parameter declaration lists must differ sufficiently such that it's possible to unambiguously determine for each <generated type reference>, which type template it references.

4.12 Row types

A row type is a sequence of (<field name> <data type>) pairs. It is described by a row type descriptor. A row type descriptor contains:

— the degree of the row type (the number of the field descriptors) and

— the field descriptor of every field of the row type.

The data type of a row of a table is a row type. In this case, every column of the row corresponds to the field of the row type that has the same ordinal position as the column.

4.13 Collection types

A collection
ANSI consists of zero or more elements.
ISO is a multiset of objects or values called elements.
All elements in a collection have the same <data type>, and each <collection type> specifies that <data type>.

A collection can be used as a <simple table>, as described in Subclause 7.14, "<query expression>".

ANSI Only–SQL3

A collection of elements may either be specified to be a general collection, not further specialized, or it may be specified to be a set, a multiset, or a list. A general collection cannot be instantiated, but it can be given the value of any set, multiset, or list of the same element type as specified for the collection.

Each element of a collection corresponds to a row in the table. That table will have a single column of that element type. A collection is a value; that is, a collection has no OID.

When a collection is used in a <simple table>, the operations defined on tables (*e.g.*, INTERSECTION) also apply to collections. Such a collection is transformed into a table for the purpose of <simple table>. Rows of that table correspond to elements in the collection. A collection type is a
ANSI general collection type,
set type, multiset type or list type.

A collection type is described by a collection type descriptor that includes:

— The name of the data type of the elements;

— An indication of whether the collection is a
ANSI general collection, or is a
set, multiset, or list.

4.13.1 Set type

A set type is a collection type that has the semantics of a mathematical set: a set is an unordered collection of distinct elements.

A set value with elements of type *DT* has the data type SET(*DT*). An empty set (a set with no elements) has the data type SET( ).

Two set values are comparable if the data types of their elements are the same or one or both set values have the data type SET( ).

Set values are assignable if the source value has the same data type as the data type of the target item or has SET( ) data type. A set value of type SET(*DT*) retains its data type even when it becomes an empty set.

Two sets are equal if they contain the same elements and those elements are all non-null values. The equality of two sets is unknown if the sets contain null values and the replacement of those null value with appropriate non-null values would make the two sets equal. Otherwise, two sets are unequal.

ANSI Only–SQL3

For any data type *DT*, a set type of *DT* elements is a direct subtype of a general collection of *DT* elements.

4.13.2 Multiset type

The multiset type is a collection type that is similar to the set type, except that duplicate elements are permitted in a multiset.

A multiset value with elements of type *DT* has the data type MULTISET(*DT*). An empty multiset (a multiset with no elements) has the data type MULTISET( ).

Two multiset values are comparable if the data types of their elements are the same or one or both multiset values have the data type MULTISET( ).

Multiset values are assignable if the source value has the same data type as the data type of the target item or has MULTISET( ) data type. A multiset value of type MULTISET(*DT*) retains its data type even when it becomes an empty multiset. Two multisets are equal if they contain the same elements and those elements are all non-null values. The equality of two multisets is unknown if the multisets contain null values and the replacement of those null value with appropriate non-null values would make the two multisets equal. Otherwise, two multisets are unequal.

ANSI Only–SQL3

For any data type *DT*, a multiset type of *DT* elements is a direct subtype of a general collection of *DT* elements.

4.13.3 List type

A list type is a collection type that is similar to the multiset type, except that the elements of a list are ordered. Thus, when an element is inserted into a list, the insert operation may specify the location in the list where the new element is to be inserted.

A list value with elements of type *DT* has the data type LIST(*DT*). An empty list (a list with no elements) has the data type LIST( ).

Two list values are comparable if the data types of their elements are the same or one or both list values have the data type LIST( ).

List values are assignable if the source value has the same data type as the data type of the target item or has LIST( ) data type. A list value of type LIST(*DT*) retains its data type even when it becomes an empty list.

Two lists are equal if they contain the same elements in the same order and those elements are all non-null values. The equality of two lists is unknown if the lists contain null values and the replacement of those null value with appropriate non-null values would make the two lists equal. Otherwise, two lists are unequal.

ANSI Only–SQL3

For any data type *DT*, a list type of *DT* elements is a direct subtype of a general collection of *DT* elements.

4.14 Row identifiers

A row identifier is an encapsulated data type that is used to identify rows of base tables in queries or in foreign references.

The value of a row identifier for a given base table row is equal to itself and it not equal to the value of a row identifier for any other base table row within the database.

Any base table may have a row identifier explicxitly defined. Any base table that has a subtable or supertable will have a row identifier implicitly defined. It is implementation-defined whether base tables without supertables or subtables have row identifiers implicitly defined.

For every table for which a row identifier is defined, a new subtype of row identifier data type is defined. A row identifier defined for a table with no supertables is a subtype of the row identifier data type. A row identifier of a table with supertables is a subtype of the row identifier data type defined for each supertable.

Every table that has a row identifier defined has an implicit column named IDENTITY, which contains the unique row identifier value (and subtype) for each row.

The term *implicit column* means that the column has no ordinal position and does not appear in ordinal position column lists (such as in an <insert statement> with no column list, or the implicit column name list corresponding to an <asterisk> in a <select list>), but may be referenced by a <column reference>.

A value corresponding to a row identifier data type *RV* is assignable to a row identifier data type *R* if *RV* is equal to *R* or if *RV* is a subtype of *R*.

The row identifier type is described by a row identifier type descriptor. A row identifier type descriptor contains:

— an indication that this is a row identifier type, and

— the name of the table within the row identifier type is used.

There is no <literal> corresponding to a row identifier.

The mapping of a row identifier value to and from supported interfaces has the properties that the mapping in both directions is unique and one-to-one. In host languages, the mapped data type is a HANDLE of a row. Consequently, such mappings are valid for the duration of the SQL-transaction or cursor that initially queried the value. Validity of such mappings for longer durations or simultaneously in different SQL-transactions is implementation-defined. The source language data type mapped to (*i.e.*, the native host language data type of a HANDLE) is implementation-defined.

4.15 Type conversions and mixing of data types

Values of the data types NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, and DOUBLE PRECISION are numbers and are all mutually comparable and mutually assignable. If an assignment would result in a loss of the most significant digits, an exception condition is raised. If least significant digits are lost, implementation-defined rounding or truncating occurs with no exception condition being raised. The rules for arithmetic are generally governed by Subclause 6.17, "<numeric value expression>".

Values corresponding to the data types CHARACTER, CHARACTER VARYING, and CHARACTER LARGE OBJECT are mutually assignable if and only if they are taken from the same character repertoire. If they are from different character repertoires, then the value of the source of the assignment must be translated to the character repertoire of the target before an assignment is possible. If a store assignment would result in the loss of non-<space> characters due to truncation, then an exception condition is raised. The values are mutually comparable only if they are mutually assignable and can be coerced to have the same collation. The comparison of two character strings depends on the collating sequence used for the comparison (see Table 3, "Collating sequence usage for comparisons"). When values of unequal length are compared, if the collating sequence for the comparison has the NO PAD attribute and the shorter value is equal to a prefix of the longer value, then the shorter value is considered less than the longer value. If the collating sequence for the comparison has the PAD SPACE attribute, for the purposes of the comparison, the shorter value is effectively extended to the length of the longer by concatenation of <space>s on the right.

Values corresponding to the binary data type are mutually assignable. If a store assignment would result in the loss of octets due to truncation, then an exception condition is raised. When binary string values are compared, they must have exactly the same length (in octets) to be considered equal. Binary string values can only be compared for equality.

Values corresponding to the data types BIT and BIT VARYING are always mutually comparable and are mutually assignable. If a store assignment would result in the loss of bits due to truncation, then an exception condition is raised. When values of unequal length are compared, if the shorter is a prefix of the longer, then the shorter is less than the longer; otherwise, the longer is effectively truncated to the length of the shorter for the purposes of comparison. When values of equal length are to be compared, then a bit-by-bit comparison is made. A 0-bit is less than a 1-bit.

Values corresponding to the data type boolean are always mutually comparable and are mutually assignable.

Values of type datetime are mutually assignable only if the source and target of the assignment have the same datetime fields.

Values of type interval are mutually assignable only if the source and target of the assignment are both year-month intervals or if they are both day-time intervals.

Values corresponding to distinct types are mutually assignable and mutually comparable if and only if both have the same distinct type.

Values of two abstract data types are mutually comparable if there is a most specific unique data type that is a supertype of both abstract data types. A value of an abstract data type ADT_i is assignable to an item of data type ADT_j if and only if ADT_i is a subtype of ADT_j.

A value corresponding to a value ADT is not comparable to a value corresponding to an object ADT.

Note: Explicit CAST functions or attribute comparisons can be used to make both values of the same subtype family or to perform the comparisons on attributes of the ADTs.

Note: “Subtype” and “subtype family” are defined in Subclause 4.11.5, "Subtypes and supertypes for ADTs".

Values corresponding to the collection types are discussed in Subclause 4.13, "Collection types".

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

ANSI Only–SQL3

Values corresponding to row identifiers are mutually comparable.

Values corresponding to row types are mutually assignable if and only if both have the same degree and every field in one row type is mutually assignable to the field in the same ordinal position of the other row type. Values corresponding to row types are mutually comparable if and only if both have the same degree and every field in one row type is mutually comparable to the field in the same ordinal position of the other row type.

4.16 Operators

An operator is applied to one or more values to return another value. Operators are either *predefined standard operators* or *user-defined operators*. When a predefined standard operator is applied to values of predefined data types, it represents the predefined operation. When a predefined standard operator is applied to one or more abstract data values, or when a user-defined operator is applied to values of primitive data types or abstract data values, the operator expression is implicitly transformed into a routine invocation.

User-defined operator definitions are associated with schemas. The operator definitions that apply to a given SQL-statement are the operator definitions associated with the default schema for that SQL-statement.

An operator is described by an operator descriptor. An operator descriptor includes:

— the name of the operator,

— type of the operator (prefix, postfix, infix),

— precedence of the operator (primary, term, factor), and

— operator symbol.

SQL supports pre-defined standard operators that are listed in Table 9, "Predefined standard operators".

Table 9—Predefined standard operators

Operator Name	Type	Precedence	Symbol	Routine Name
PLUS	Prefix	Primary	+	PLUS
MINUS	Prefix	Primary	-	MINUS
MULTIPLY	Infix	Factor	*	MULTIPLY
DIVIDE	Infix	Factor	/	DIVIDE
ADD	Infix	Term	+	ADD
SUBTRACT	Infix	Term	-	SUBTRACT
CONCATENATE	Infix	Factor	||	CONCATENATE

4.17 Domains

A domain is a set of permissible values. A domain is defined in a schema and is identified by a <domain name>. The purpose of a domain is to constrain the set of valid values that can be stored in SQL-data by various operations.

A domain definition specifies a data type. It may also specify a <domain constraint> that further restricts the valid values of the domain and a <default clause> that specifies the value to be used in the absence of an explicitly specified value or column default.

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

— the name of the domain;

— the data type descriptor of the data type of the domain;

— the <collation name> from the <collate clause>, if any, of the domain;

— the null class of the domain;

— the value of <default option>, if any, of the domain; and

— the domain constraint descriptors of the domain constraints, if any, of the domain.

4.18 Nulls

Every domain, column in a base table, SQL variable, and SQL-supplied parameter, has a null class. If no null class is specified, it is the general null class (which contains only the general null value); otherwise it is the defined null class that is specified.

A defined null class, is created by a <null class definition> and is a named set of possible null values known as null states, together with the general null value. A null state is a named, implementation-dependent null value that is distinct from both the general null value and all other null states of the same null class.

The null values of a null class are ordered on their position number, the general null value having position number one in every null class. This ordering is used to determine the result of an operation when more than one of its operands are null values.

Except in the few cases where the null substitution principle yields a different result, if one or more operands of an expression is a null value then the result is a null value.

The result of an attempted transfer of any null value between objects having different null classes is the general null value.

The null class of the result of an operation is determined as follows.

Case:

— If no operand is null, then the result is as determined by the application of other General Rules.

— If the operator is AND and either operand is false, then the result is false.

— If the operator is OR and either operand is true, then the result is true.

— If the result has a defined null class, and one or more operands are null, then:

  Case:

  * If the operator is OR then the result is the null value having the maximum position number;

  * Otherwise, the result is the null value having the minimum position number.
    Note: The general null value effectively has the position number one.

— Otherwise, the result is the general null value.

4.19 Columns, fields, and attributes

A column is a multiset of values that may vary over time. All values of the same column are of the same data type or domain and are values in the same table. A value of a column is the smallest unit of data that can be selected from a table and the smallest unit of data that can be updated. If a table is a collection of instances of an abstract data type, then a column of the table is a multiset of the values of an attribute of that ADT and has the name of that attribute.

Every column has a <column name>.

Every column and attribute has a nullability characteristic that indicates whether any attempt to store a null value into that column or attribute will inevitably cause an exception to be raised, and whether any attempt to retrieve a value from that column or attribute can ever result in a null value. The possible values of the nullability characteristic are *known not nullable* and *possibly nullable*.

A column *C* with <column name> *CN* of a base table *T* has a nullability characteristic that is *known not nullable* if and only if either:

— there exists at least one constraint that is not deferrable and that simply contains a <search condition> that contains *CN* IS NOT NULL or NOT *CN* IS NULL or *RVE* IS NOT NULL, where *RVE* is a <row value constructor> that contains a <row value constructor element> that is simply *CN* without an intervening <search condition> that specifies OR and without an intervening <boolean factor> that specifies NOT.

— *C* is based on a domain that has a domain constraint that is not deferrable and that simply contains a <search condition> that contains VALUE IS NOT NULL or NOT VALUE IS NULL without an intervening <search condition> that specifies OR and without an intervening <boolean factor> that specifies NOT.

— *C* is a unique column of a nondeferrable unique constraint that is a PRIMARY KEY.

An attribute *A* of an ADT is *known not nullable* if and only if *A* is based on a domain that has a domain constraint that is not deferrable and that simply contains a <search condition> that contains VALUE IS NOT NULL or NOT VALUE IS NULL without an intervening <search condition> that specifies OR and without an intervening <boolean factor> that specifies NOT.

****Editor's Note****
During Munich consideration of MUN-122/X3H2-93-488, it was noted that "known not nullable" is not a useful concept when applied to attributes. See Possible Problem **337** in the Editor's Notes.

Otherwise, a column or attribute *C* is *possibly nullable*.

A column, *C*, is described by a column descriptor. A column descriptor includes:

— the name of the column;

— whether the name of the column is an implementation-dependent name;

— if the column is based on a domain, then the name of that domain; otherwise, the data type descriptor of the data type of *C* and the null class of *C*;

— the <collation name> from the <collate clause>, if any, of *C*;

— the value of <default option>, if any, of *C*;

— the nullability characteristic of *C*; and

— the ordinal position of *C* within the table that contains it.

****Editor's Note****
During Munich consideration of MUN-081/X3H2-93-483, it was noted that column descriptors do not have the original name of an inherited column in the supertable. See Possible Problem **323** in the Editor's Notes.

An attribute *A* is described by an attribute descriptor. An attribute descriptor includes:

— the name of the attribute;

— if the attribute is based on a domain, then the name of that domain; otherwise, the data type descriptor of the data type of *A* and the null class of *A*;

— the <collation name> from the <collate clause>, if any, of *A*;

— the value of <default option>, if any, of *A*;

— the nullability characteristic of *A*;

• 2 list elements deleted

— An indication of whether the attribute is UPDATABLE, CONSTANT, or READ ONLY;

— the ordinal position of *A* within the abstract data type that contains it; and

— the encapsulation level of *A*.

A field *F* is described by a field descriptor. A field descriptor includes:

— the name of the field;

— if the field is based on a domain, then the name of that domain; otherwise, the data type descriptor of the data type of *F* and the null class of *F*;

— the <collation name> from the <collate clause>, if any, of *F*;

— the ordinal position of *F* within the row type that smply contains it.

4.20 Tables

A table is collection of rows. A row is an instance of a row type. Every row of the same table has the same row type. The value of the *i*-th field of every row in a table is the value of the *i*-th column of that row in the table. The row is the smallest unit of data that can be inserted into a table and deleted from a table.

ANSI Only–SQL3

Since a table is a collection of rows, a collection can be used in any context where a table expression is permitted, by explicitly converting the collection into a table with a row of one value (as specified in Subclause 7.14, "<query expression>"). Just as every collection is either a set, multiset, or list, so every table is either a set table, multiset table, or list table. Set tables and list tables share all of the properties of multiset tables (the default), but have the additional properties that:

— A set table can contain no duplicate rows; and

— A list table has an ordering of the rows of the table.

The data type of a table specifies

— whether the table is a multiset table, set table, or list table; and

— the row type of the table.

• 1 paragraph deleted

The degree of a table is the number of columns of that table. At any time, the degree of a table is the same as the cardinality of each of its rows and the cardinality of a table is the same as the cardinality of each of its columns. A table whose cardinality is 0 is said to be *empty*.

A table is either a base table or a derived 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.

A viewed table is either a permanent viewed table or a temporary viewed table. A permanent viewed table is called a *view* or a *viewed table*; a temporary viewed table is called a *temporary view*.

A persistent base table is a named table defined by a <table definition> that does not specify TEMPORARY.

A derived table is a table derived directly or indirectly from one or more other tables by the evaluation of a <query expression>. The values of a derived table are derived from the values of the underlying tables when the <query expression> is evaluated.

A viewed table is a named derived table defined by a <view definition>. A viewed table is sometimes called a *view*.

The terms *simply underlying table*, *underlying table*, *leaf underlying table*, *generally underlying table*, and *leaf generally underlying table* define a relationship between a derived table or cursor and other tables.

The *simply underlying table*s of derived tables and cursors are defined in Subclause 7.13, "<query specification>", Subclause 7.14, "<query expression>", and Subclause 13.1, "<declare cursor>". A viewed table has no *simply underlying table*s.

The *underlying table*s of a derived table or cursor are the simply underlying tables of the derived table or cursor and the underlying tables of the simply underlying tables of the derived table or cursor.

The *leaf underlying table*s of a derived table or cursor are the underlying tables of the derived table or cursor that do not themselves have any underlying tables.

The *generally underlying table*s of a derived table or cursor are the underlying tables of the derived table or cursor and, for those underlying tables of the derived table or cursor that are viewed tables, the <query expression> of each viewed table and the generally underlying tables of the <query expression> of each viewed table.

The *leaf generally underlying table*s of a derived table or cursor are the generally underlying tables of the derived table or cursor that do not themselves have any generally underlying tables.

All base tables are inherently updatable. Derived tables are either inherently updatable or not inherently updatable. The operations of insert, update, and delete are permitted for inherently updatable tables, subject to constraining Access Rules. These operations may also be permitted for tables that are not inherently updatable if appropriate triggers have been defined.

A grouped table is a set of groups derived during the evaluation of a <group by clause> or a <having clause>. A group is a multiset of rows in which all values of the grouping column or columns are equal if a <group by clause> is specified, or the group is the entire table if no <group by clause> is specified. A grouped table may be considered as a collection of tables. Set functions may operate on the individual tables within the grouped table.

A global temporary table is a named table defined by a <table definition> that specifies GLOBAL TEMPORARY. A created local temporary table is a named table defined by a <table definition> that specifies LOCAL TEMPORARY. Global and created local temporary tables are effectively materialized only when referenced in an SQL-session. Every <module> in every SQL-session that references a created local temporary table causes a distinct instance of that created local temporary table to be materialized. That is, the contents of a global temporary table or a created local temporary table cannot be shared between SQL-sessions. In addition, the contents of a created local temporary table cannot be shared between <module>s of a single SQL-session. The definition of a global temporary table or a created local temporary table appears in a schema. In SQL language, the name and the scope of the name of a global temporary table or a created local temporary table are indistinguishable from those of a persistent base table. However, because global temporary table contents are distinct within SQL-sessions, and created local temporary tables are distinct within <module>s within SQL-sessions, the *effective* <schema name> of the schema in which the global temporary table or the created local temporary table is instantiated is an implementation-dependent <schema

name> that may be thought of as having been effectively derived from the <schema name> of the schema in which the global temporary table or created local temporary table is defined and the implementation-dependent SQL-session identifier associated with the SQL-session. In addition, the *effective* <schema name> of the schema in which the created local temporary table is instantiated may be thought of as being further qualified by a unique implementation-dependent name associated with the <module> in which the created local temporary table is referenced.

A declared local temporary table is either a module local temporary table or a compound statement local temporary table. A module local temporary table is a named table defined by a <temporary table declaration> in a <module>. A module local temporary table is effectively materialized the first time it is referenced in an SQL-session, and it persists for that SQL-session.

The materialization of a temporary table does not persist beyond the end of the SQL-session in which the table was materialized. Temporary tables are effectively empty at the start of an SQL-session.

A subtable is a named table defined by a <table definition> that contains a <subtable clause>. If table T_a is declared to be a subtable of another table T_b by a <table definition>, T_a is called a *direct subtable* of T_b. The subtable is defined as follows:

Table T_a is a subtable of T_b if one of the following holds,

1) T_a is a direct subtable of T_b.

2) There is a table T_c such that T_a is a direct subtable of T_c and T_c is a subtable of T_b.

Table T_b is called a supertable of T_a if T_a is a subtable of T_b. If T_a is a direct subtable of T_b, then T_b is called a *direct supertable* of T_a. A supertable that is not a subtable of any other table is called a *maximal supertable*. A supertable shall not have itself as a subtable.

A subtable is a specialized table of one or more supertables, and a supertable is a generalized table of one or more subtables. Supertables and subtables are base tables. Supertables and subtables may have the same rows.

When a subtable is defined, the subtable inherits every column from its supertables. The phrase *inherited column* denotes a column that is inherited from some supertable. The phrase *originally-defined column* denotes a column that is specified in the <table definition> of the subtable.

Let T_a be a maximal supertable and T be a subtable of T_a. The set T_a and all subtables of T_a is called a *subtable family* of T or of T_a. Given a row of a maximal supertable T_a, the set of rows which each have the same primary key value and belong to a table in the corresponding subtable family is called a *subtable row family*. There shall be only one maximal supertable in a subtable family.

Any row of a subtable must correspond to one and only one row of each direct supertable. Any row of a supertable corresponds to at most one row of a direct subtable. Two column values of two rows belonging to the same subtable row family coincide if the two columns are inherited from the same column.

A temporary viewed table is a named derived table defined by a <temporary view declaration>. A temporary view does not persist across SQL-sessions. A <temporary view declaration> appears within a <module>. Whether a temporary viewed table is materialized is implementation-dependent.

A table is described by a table descriptor. A table descriptor is either a base table descriptor, a view descriptor, or a derived table descriptor (for a derived table that is not a view).

Every table descriptor includes:

ANSI Only–SQL3

— an indication of whether the table is a multiset table, a set table, or a list table;

— the degree of the table (the number of column descriptors); and

— the column descriptor of each column in the table.

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

— the name of the base table;

— an indication of whether the table is a persistent base table, a global temporary table, a created local temporary table, or a declared local temporary table;

— the descriptor of each table constraint specified for the table;

— the row type of the table;

• 1 entry deleted.

— if the table is a subtable, then the names of its direct supertables;

— a non-empty set of functional dependencies, according to the rules given in Subclause 4.22, "Functional dependencies and candidate keys";

— a non-empty set of candidate keys; and

— a preferred candidate key, which may or may not be additionally designated the primary key, according to the Rules in Subclause 4.22, "Functional dependencies and candidate keys".

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

— the <query expression> that defines how the table is to be derived; and

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

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

— the name of the view;

— an indication of whether the view has the CHECK OPTION; if so, whether it is to be applied as CASCADED or LOCAL; and

— an indication of whether it is a persistent view or a temporary view.

4.21 Integrity constraints

Integrity constraints, generally referred to simply as constraints, define the valid states of SQL-data by constraining the values in the base tables. A constraint is either a table constraint, a domain constraint or an assertion. A constraint is described by a constraint descriptor. A constraint descriptor is either a table constraint descriptor, a domain constraint descriptor or an assertion descriptor. Every constraint descriptor includes:

— the name of the constraint;

— an indication of whether or not the constraint is deferrable;

— an indication of whether the initial constraint mode is *deferred* or *immediate*;

A <query expression> or <query specification> is *possibly non-deterministic* if an implementation might, at two different times where the state of the SQL-data is the same, produce results that differ by more than the order of the rows due to General Rules that specify implementation-dependent behavior.

No integrity constraint shall be defined using a <query specification> or a <query expression> that is possibly non-deterministic.

4.21.1 Checking of constraints

Every constraint is either *deferrable* or *non-deferrable*. Within an SQL-transaction, every constraint has a constraint mode; if a constraint is *non-deferrable*, 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 may be changed (from *immediate* to *deferred*, or from *deferred* to *immediate*) by execution of a <set constraints mode statement>.

ANSI Only–SQL3

An SQL-statement *S1* is said to *immediately cause* the execution of another SQL-statement *S2* if the target of *S1* is some table *T*, *T* is named by the <table name> of a <trigger definition> *TD*, and *TD* contains a <triggered SQL statement> that generally contains *S2*. An SQL-statement *S1* is said to *cause* the execution of another SQL-statement *S3* if *S* immediately causes *S3* or if *S* immediately causes some SQL-statement *S2* that causes *S3*. An SQL-statement *S1* is said to be the *root cause* of the execution of some other SQL-statement *S2* if *S1* causes *S2* and *S* is not caused by any SQL-statement.

The checking of a constraint depends on its constraint mode within the current SQL-transaction. If the constraint mode is *immediate*, then the constraint is effectively checked at the end of each

ANSI Only–SQL3

SQL-statement *S*, unless *S* is executed because it is a <triggered SQL statement>, in which case, the constraint is effectively checked at the end of the SQL-statement that is the root cause of *S*.

ISO Only—caused by ANSI changes not yet considered by ISO

SQL-statement.

If the constraint mode is *deferred*, then the constraint is effectively checked when the constraint mode is changed to *immediate* either explicitly by execution of a <set constraints mode statement>, or implicitly at the end of the current SQL-transaction.

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

4.21.2 Table constraints

A table constraint is either a unique constraint, a referential constraint or a table check constraint. A table constraint is described by a table constraint descriptor which is either a unique constraint descriptor, a referential constraint descriptor or a table check constraint descriptor.

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

— an indication of whether it was defined with PRIMARY KEY or UNIQUE; and

— the names and positions of the *unique columns* specified in the <unique column list>.

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

— the names of the *referencing columns* specified in the <referencing columns>;

— the names of the *referenced columns* and *referenced table* specified in the <referenced table and columns>;

— an indication of whether PENDANT was specified; and

— the value of the <match type>, if specified, and the <referential triggered actions>, if specified.

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

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

— the <search condition>.

A unique constraint is satisfied if and only if no two rows in a table have the same non-null values in the *unique columns*. In addition, if the unique constraint was defined with PRIMARY KEY, then it requires that none of the values in the specified column or columns be a null value.

In the case that a table constraint is a referential constraint, the table is referred to as the *referencing table*. The *referenced columns* of a referential constraint shall be the *unique columns* of some unique constraint of the *referenced table*.

A referential constraint is satisfied if one of the following conditions is true, depending on the <match option> specified in the <referential constraint definition>:

— If no <match type> was specified then, for each row *R1* of the *referencing table*, either at least one of the values of the *referencing columns* in *R1* shall be a null value, or the value of each referencing column in *R1* shall be equal to the value of the corresponding *referenced column* in some row of the *referenced table*.

— If MATCH FULL was specified then, for each row *R1* of the *referencing table*, either the value of every *referencing column* in *R1* shall be a null value, or the value of every *referencing column* in *R1* shall not be null and there shall be some row *R2* of the *referenced table* such that the value of each *referencing column* in *R1* is equal to the value of the corresponding *referenced column* in *R2*.

— If MATCH PARTIAL was specified then, for each row *R1* of the *referencing table*, there shall be some row *R2* of the *referenced table* such that the value of each *referencing column* in *R1* is either null or is equal to the value of the corresponding *referenced column* in *R2*.

The referencing table may be the same table as the referenced table.

A table check constraint is satisfied if and only if the specified <search condition> is not false for any row of a table.

A PENDANT specification further requires that for each row of the *referenced table*, the values of the specified column or columns are the same as the values of the specified column or columns in some row of some *referencing table*.

4.21.3 Domain constraints

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 <search condition>.

A domain constraint is satisfied by SQL-data if and only if, for any table *T* that has a column named *C* based on that domain, the specified <search condition>, with each occurrence of VALUE replaced by *C*, is not false for any row of *T*.

A domain constraint is satisfied by the result of a <cast specification> if and only if the specified <search condition>, with each occurrence of VALUE replaced by that result, is not false.

4.21.4 Assertions

An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables.

An assertion is described by an assertion descriptor. In addition to the components of every constraint descriptor an assertion descriptor includes:

— the <search condition>.

An assertion is satisfied if and only if the specified <search condition> is not false.

4.22 Functional dependencies and candidate keys

A *functional dependency* is a truth-valued statement pertaining to a table. If *A* and *B* are both arbitrary subsets (possibly empty) of the columns of a table *T*, then:

$A \mapsto B$

means "*A determines B*", or "*B is functionally dependent on A*". If the statement holds true in *T*, then any two rows of *T* that agree in value in every column in *A* also agree in value in every column in *B*. The set of columns *A* is said to be a *determinant of* the set of columns *B* and the members of *B* are said to be *dependants of A*.

Associated with any table *T* is a non-empty set of functional dependencies that are constrained to hold true over time, for any value of *T*. For example, if *PK* is the set of columns comprising the defined primary key of some base table *BT*, and *CT* is the set comprising all the columns of *BT*, then the functional dependency $PK \mapsto CT$ holds true over time in *BT*. The set of functional dependencies is never empty, because, if *CT* is the set comprising all the columns of some table *T*, then $CT \mapsto CT$ always holds true, trivially, in *T*.

If the functional dependency $CK \mapsto CT$ holds true in some table *T*, where *CT* comprises all the columns of *T*, and there is no proper subset *CK1* of *CK* such that $CK1 \mapsto CT$ holds true in *T*, then *CK* is a *candidate key* of *T*. Note that if no proper subset of *CT* is a candidate key, then *CT* is the only candidate key of *T*.

This Subclause defines functional dependency and candidate key and specifies a minimal set of rules that a conforming implementation must follow to determine functional dependencies and candidate keys in base tables and <query expression>s.

The rules in this Subclause, though sufficient for most practical purposes, do not guarantee always to give minimal candidate keys. For this reason, they may be freely augmented by implementation-defined rules, where indicated in this Subclause.

This Subclause also specifies a rule whereby one candidate key is noted as the preferred candidate key of any <query expression>. In some cases, this is also designated as the *primary key*.

Other Clauses of this international standard may specify rules in connection with the functional dependencies, candidate keys, preferred candidate keys and primary keys specified by this Subclause.

4.22.1 General rules and definitions

Let *T* be any table. Let *CT* be the set comprising all the columns of *T*, and let *A* and *B* be arbitrary subsets of *CT*, not necessarily disjoint and possibly empty.

Let $A \mapsto B$ denote the functional dependency of *B* on *A* in *T* holding true if, for any possible value of *T*, any two rows that agree in value for every column in *A* also agree in value for every column in *B*. Two rows agree in value for a column if the two values either are both null or compare as equal under the General Rules of Subclause 8.2, "<comparison predicate>". Let the phrase "$A \mapsto B$ is a functional dependency in *T*" denote that $A \mapsto B$ holds true for any possible value of *T*.

If $X \mapsto Y$ is some functional dependency in some table *T*, then *X* is a determinant in *T*.

Let $C1 \mapsto B$ denote that *B* is functionally dependent on the set containing exactly one column, *C1*. Let $A \mapsto C2$, for example, denote that the single column *C2* is functionally dependent on the set of columns *A*.

Let $A \mapsto B$ and $C \mapsto D$ be any two functional dependencies in *T*. The following are also functional dependencies in *T*:

— *A* UNION (*C* DIFFERENCE *B*) $\mapsto$ *B* UNION *D*

— *C* UNION (*A* DIFFERENCE *D*) $\mapsto$ *B* UNION *D*

The term *candidate key* is defined as follows:

> If $K1 \mapsto CT$ is a functional dependency in *T* and there is no subset *K2* of *K1* such that $K2 \mapsto CT$ is also a functional dependency in *T*, then *K1* is a candidate key of *T*.

Every table has an associated non-empty set of functional dependencies and a non-empty set of candidate keys.

The set of functional dependencies is non-empty because $X \mapsto X$ for any *X*. A functional dependency of this form is an axiomatic functional dependency, as is $X \mapsto Y$ where *Y* is a subset of *X*. $X \mapsto Y$ is a non-axiomatic functional dependency if *Y* is not a subset of *X*.

The set of candidate keys is non-empty because, if no proper subset of *CT* is a candidate key, then *CT* is a candidate key.

Let Φ denote the empty set.

Let *BPK-member* denote a property of a column, such that "*C1* is *BPK-member*" and "*C2* is not *BPK-member*" can be written. A column *C3* is *BPK-member* if it is a unique column of some unique constraint *UC* of some table, and *UC* specifies PRIMARY KEY; otherwise *C3* is not *BPK-member*.

Let a column *C1* be a *counterpart* of a column *C2* if *C1* is specified by a column reference (or by a <value expression> that is a column reference) that references *C2*. If *C2* is a counterpart of *C1* and *C3* is a counterpart of *C2*, then *C3* is a counterpart of *C1*.

4.22.2 Functional dependencies in a base table

Let *T* be a base table and let *CT* be the set comprising all the columns of *T*.

A column *C1* of *T* is or is not *BPK-member*, in accordance with the General Rules of Subclause 4.22.1, "General rules and definitions".

If the table descriptor of *T* includes a table constraint descriptor that indicates that the constraint was defined with PRIMARY KEY and is not deferrable, then let *UCL* be the <unique column list> of the primary key of *T*; otherwise let *UCL* be *CT*.

$UCL \mapsto CT$ is a functional dependency in T.

Implementation-defined rules may determine other functional dependencies in *T*.

4.22.3 Functional dependencies in <table value constructor>

Let *R* be the result of a <table value constructor>, and let *CR* be the set comprising all the columns of *R*.

No column of *R* is *BPK-member*.

If *R* directly contains exactly one <row value constructor>, then $\Phi \mapsto CR$ is a functional dependency in *R*; otherwise $CR \mapsto CR$ is the only functional dependency in *R*.

4.22.4 Functional dependencies in a <joined table>

Let *T1* and *T2* denote the tables identified by the first and second <table reference>s of some <joined table> *JT*. Let *R* denote the table that is the result of *JT*.

Every column of *R* has some counterpart in either *T1* or *T2*. If NATURAL is specified or the <join specification> is a <named columns join>, then some columns of *R* may have counterparts in both *T1* and *T2*.

A column *C1* of *R* is *BPK-member* if *C1* has some counterpart in *T1* or *T2* that is *BPK-member*; otherwise *C1* is not *BPK-member*.

Case:

— If the <join type> is UNION, then no non-axiomatic functional dependency in *T1* or *T2* is a functional dependency in *R*.

— If the <join type> is FULL, then no non-axiomatic functional dependency in *T1* or *T2* whose determinant includes any column that is not *BPK-member* is a functional dependency in *R*.

— If the <join type> is LEFT, then no non-axiomatic functional dependency in *T2* whose determinant includes any column that is not *BPK-member* is a functional dependency in *R*.

— If the <join type> is RIGHT, then no non-axiomatic functional dependency in *T1* whose determinant includes any column that is not *BPK-member* is a functional dependency in *R*.

Except for those discarded under the previous rule, all functional dependencies in *T1* and *T2* are functional dependencies in *R*.

Case:

— If a <cross join> is specified, then there are no further functional dependencies in *R*.

— If a <join condition> is specified and *SC* is the <search condition> directly contained in that <join condition>, then any functional dependencies in the <query expression>

  SELECT * FROM *T1*, *T2* WHERE *SC*

  are also functional dependencies in *R*.

— Otherwise, further functional dependencies in *R* are determined by treatment of equivalent <query expression>s given in Subclause 7.9, "<joined table>", always taking the <join type> as INNER (even if an <outer join type> is actually given).

4.22.5 Functional dependencies in the result of a <from clause>

Let *R* be the result of some <from clause> *FC*.

If *T* is a <table reference> directly contained in *FC*, then all functional dependencies in *T* are functional dependencies in *R*.

4.22.6 Functional dependencies in the result of a <where clause>

Let *T* be the table that is the operand of the <where clause>. Let *R* be the result of the <where clause>.

Every column in *R* has a counterpart in *T*. A column *C1* in *R* is *BPK-member* if its counterpart in *T* is *BPK-member*; otherwise *C1* is not *BPK-member*.

All functional dependencies in *T* are functional dependencies in *R*.

Let *T1* be *T*, extended to the right with columns arising from <value expression>s contained in the <search condition>, as follows:

Case:

— A <value expression> that is a <literal>, a <general value specification>, or a <datetime value function> is a constant column in *T1*.

— A <value expression> *VE* that is not a column reference specifies a computed column *CC* in *T1*. For every row in *T1*, the value in *CC* is the result of *VE*.

If *CONST* is some constant column in *T1*, then $\Phi \mapsto CONST$ is a functional dependency in *T1*.

If *CC* is some computed column in *T1*, then let *OP1*, *OP2*, . . . be the operands of the <value expression> *VE* that specifies *CC*.

$\{OP1, OP2, \ldots\} \mapsto CC$ is a functional dependency in *T1*.

Further functional dependencies may be determined by implementation-defined rules.

If the <search condition> constrains some column *C1* of *T1* to be equal to some other column *C2* of *T1*, then $C1 \mapsto C2$ and $C2 \mapsto C1$ are functional dependencies in *T1*.

Note: A conforming implementation is required minimally to detect equality constraints implied by simple sub-expressions of the form *C1* = *C2*, not negated and not unconstrained by the use of "OR", where *C1* and *C2* are columns of *T1*.

4.22.7 Functional dependencies in the result of a <group by clause>

Let *T1* be the table that is the operand of the <group by clause>, and let *R* be the result of the <group by clause>.

Let *G* be the set of columns specified by the <collated column reference list> of the <group by clause>.

Note: A <query specification> that does not directly contain a <group by clause>, but whose directly contained <select list> includes a <set function specification>, is treated by this Subclause, with the empty set as *G*.

The columns of *R* are the columns of *G*, with an additional column *CI*, whose value in any particular row of *R* somehow denotes the subset of rows of *T1* that is associated with the combined value of the columns of *G* in that row.

Any column *GC* of *G* has a counterpart in *T1*. *GC* is *BPK-member* if its counterpart in *T1* is *BPK-member*; otherwise, *GC* is not *BPK-member*.

$G \mapsto CI$ is a functional dependency in *R*.

Note: Any <set function specification> that is specified in conjunction with *R* is necessarily a function of *CI*. If *SFVC* denotes the column containing the results of such a <set function specification>, then $CI \mapsto SFVC$ holds true, and it follows that $G \mapsto SFVC$ is a functional dependency in the table containing *SFVC*.

4.22.8 Functional dependencies in the result of a <having clause>

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

Any functional dependency in the <query expression>

SELECT * FROM *T1* WHERE *SC*

is a functional dependency in *R*.

4.22.9 Functional dependencies in a <query specification>

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

Let *S*L be the <select list> of the <query specification>.

Let *T1* be *T* extended to the right with columns arising from <value expression>s contained in the <select list>, as follows.

Case:

— A <value expression> that is a <literal>, a <general value specification>, or a <datetime value function> is a constant column in *T1*.

— A <value expression> *VE* that is not a column reference specifies a computed column *CC* in *T1*. For every row in *T1*, the value in *CC* is the result of *VE*.

Let *dcr* be a column of *R* that arises from the use of * in *SL* or by the specification of a column reference as a <value expression> directly contained in *SL*. *DCR* has counterparts in *T* and *T1*. If the counterpart of *DCR* in *T* is *BPK-member*, then *dcr* is *BPK-member*; otherwise *dcr* is not *BPK-member*.

If $A \mapsto B$ is some functional dependency in *T*, then $A \mapsto B$ is a functional dependency in *T1*.

Let *CC* be the column specified by some <value expression> *VE* in the <select list>. Any functional dependency in the <query expression>

SELECT * FROM *T* WHERE *VE* = *VE*

is a functional dependency in *T1*.

Furthermore, if *OP1, OP2, . . .* are the operands of *VE*, then $\{OP1, OP2, \ldots\} \mapsto CC$ is a functional dependency in *T1*.

Let $C \mapsto D$ be some functional dependency in *T1*. If all the columns of *C* have counterparts in *R*, then let *DR* be the set comprising those columns of *D* that have counterparts in *R*. $C \mapsto DR$ is a functional dependency in *R*.

4.22.10 Functional dependencies in a <query expression>

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

If the <query expression> specifies UNION, EXCEPT or INTERSECT, then let *T1* and *T2* be the left and right operand tables and let *R* be the result. Let *CR* be the set comprising all the columns of *R*.

Each column of *R* has a counterpart in *T1* and a counterpart in *T2*.

Case:

— If EXCEPT is specified, then a column *CR* of *R* is *BPK-member* if its counterpart in *T1* is *BPK-member*; otherwise, *CR* is not *BPK-member*.

— If UNION is specified, then a column *CR* of *R* is *BPK-member* if its counterparts in *T1* and *T2* are both *BPK-member*; otherwise *CR* is not *BPK-member*.

— If INTERSECT is specified, then a column *CR* of *R* is *BPK-member* if either of its counterparts in *T1* and *T2* is *BPK-member*; otherwise *CR* is not *BPK-member*.

Case:

— If UNION is specified, then no non-axiomatic functional dependency in *T1* or *T2* is a functional dependency in *R*, apart from any functional dependencies determined by implementation-defined rules.

— If EXCEPT is specified, then all functional dependencies in *T1* are functional dependencies in *R*.

— If INTERSECT is specified, then all functional dependencies in *T1* and all functional dependencies in *T2* are functional dependencies in *R*.

Note: Other functional dependencies may be determined according to implementation-defined rules.

4.22.11 Functional dependencies in a <recursive union>

To Be Provided!

4.22.12 Selection of primary key or preferred candidate key

Let *CK* be the set of candidate keys of a table *T*.

Note: Because a candidate key is a set (of columns), *CK* is therefore a set of sets (of columns).

Let *PCK* be the set: *P* such that *P* is a member of *CK* and every member (column) of *P* is *BPK-member*.

Case:

— If *PCK* is not empty, then some member of *PCK* is the primary key of *T*. If *PCK* has more than one member, then the primary key is chosen according to the "left-most" rule, below.

— Otherwise, some member of *CK* is the preferred candidate key of *T*. If *CK* has more than one member, then the preferred candidate key is chosen according to the "left-most" rule, below.

— The "left-most" rule:

This rule uses the ordering of the columns of a table, as specified elsewhere in this standard.

To determine the left-most of two sets of columns of *T*, first list each set in the order of the column-numbers of its members, extending the shorter list with zeros to the length of the longer list. Then, starting at the left of each ordered list, step forward until a pair of unequal column numbers, one from the same position in each list, is found. The list

containing the number that is the smaller member of this pair identifies the left-most of the two sets of columns of *T*.

To determine the left-most of more than two sets of columns of *T*, take the left-most of any two sets, then pair that with one of the remaining sets and take the left-most, and so on until there are no remaining sets.

4.23 Triggers

****Editor's Note****

I have added a placeholder for some Concepts about triggers, which seems rather desirable. The prime purpose is to make the existing Subclause into a sub-sub-clause called "Triggered actions" as 4.x.1 so that the new sub-sub-clause proposed by X3H2-92-210/CBR-054, part 1 can be 4.x.2, thus not violating the ISO and ANSI prohibitions against having a .1 if there's not at least a .2. Of course, you are all welcome to propose text for this new Subclause 4.23, "Triggers"!

A trigger is defined by a <trigger definition>. A <trigger definition> specifies a trigger that is described by a trigger descriptor . A trigger descriptor includes:

— The name of the trigger;

— The name of the table that is the subject table of the trigger;

— The trigger action time
ANSI (BEFORE, INSTEAD OF, or AFTER)
ISO (BEFORE or AFTER)
of the trigger;

— The trigger event (INSERT, DELETE, or UPDATE) of the trigger;

— The old values correlation name, if any, of the trigger;

— The new values correlation name, if any, of the trigger;

— All of the triggered actions of the trigger;

— If the trigger event is UPDATE, then the trigger column list for the trigger event of the trigger, as well as an indication of whether the trigger column list was explicit or implicit;

4.23.1 Triggered actions

ISO Only–SQL3

A schema may include the definition of SQL-statements that are to be executed before or after a row is inserted into or deleted from a table or executed during the update of a row of a table. The execution of such a triggered action resulting from the insertion, deletion, or updating of a row in a table may cause the triggering of further triggered actions in other affected tables.

ANSI Only—caused by ISO changes not yet considered by ANSI

A schema may include the definition of a triggered action specifying SQL-statements that are to be executed (either once for each row or once for the whole triggering INSERT, DELETE, or UPDATE statement) before, instead of, or after rows are inserted into a table, rows are deleted from a table, or one or more columns are updated in rows of a table. The execution of such a triggered action resulting from the insertion, deletion, or updating of a row in a table may cause the triggering of further triggered actions in other affected tables.

4.23.2 Execution of triggered actions

The execution of triggered actions depends on the cursor mode of the current SQL-transaction. If the cursor mode is set to *cascade off*, then the execution of the <triggered SQL statement>s is effectively deferred until enacted implicitly by execution of a <commit statement> or a <close statement>. Otherwise, the <triggered SQL statement>s are effectively executed
ANSI before, instead of, or after
ISO either before or after
the execution of each SQL-statement, as determined by the specified <trigger action time>.

When the <triggered SQL statement>s are executed other than at the end of an SQL-statement, if their execution is not successful, then an exception condition is raised and the SQL-statement that caused the trigger to be executed has no effect other than entering exception information into the diagnostics area. When a <commit statement> implicitly causes a <triggered SQL statement> to be executed and the execution of the <triggered SQL statement> is not successful, then an exception condition is raised and the SQL-transaction is terminated by an implicit <rollback statement>.

4.24 SQL-schemas

An SQL-schema is a persistent descriptor that includes:

— the <schema name> of the SQL-schema;

— the <authorization identifier> of the owner of the SQL-schema;

— the <character set name> of the default character set for the SQL-schema;

— the <schema path specification> defining the SQL-path for SQL-invoked routines for the SQL-schema; and

— the descriptor of every component of the SQL-schema.

In this
ANSI American
ISO International
standard, the term “schema” is used only in the sense of SQL-schema. Each component descriptor is either a domain descriptor, a base table descriptor, a view descriptor, a constraint descriptor, a privilege descriptor, a character set descriptor, a collation descriptor, a translation descriptor, an abstract data type descriptor, a type template descriptor, or a routine descriptor. The persistent objects described by the descriptors are said to be *owned by* or to have been *created by* the <authorization identifier> of the schema.

A schema is created initially using a <schema definition> and may be subsequently modified incrementally over time by the execution of <SQL schema statement>s. <schema name>s are unique within a catalog.

A <schema name> is explicitly or implicitly qualified by a <catalog name> that identifies a catalog.

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

If a reference to a <table name> does not explicitly contain a <schema name>, then a specific <schema name> is implied. The particular <schema name> associated with such a <table name> depends on the context in which the <table name> appears and is governed by the rules for <schema qualified name>.

If a reference to an SQL-invoked routine that is contained in a <routine invocation> does not explicitly contain a <schema name>, then the SQL-invoked routine is selected from the default SQL-path of the schema.

4.25 Catalogs

Catalogs are named collections of schemas in an SQL-environment. An SQL-environment contains zero or more catalogs. A catalog contains one or more schemas, but always contains a schema named INFORMATION_SCHEMA that contains the views and domains of the Information Schema. The method of creation and destruction of catalogs is implementation-defined. The set of catalogs that can be referenced in any SQL-statement, during any particular SQL-transaction, or during the course of an SQL-session is also implementation-defined. The default catalog for a <module> whose <module authorization clause> does not specify an explicit <catalog name> to qualify the <schema name> is implementation-defined.

4.26 Clusters of catalogs

A cluster is an implementation-defined collection of catalogs. Exactly one cluster is associated with an SQL-session and it defines the totality of the SQL-data that is available to that SQL-session.

An instance of a cluster is described by an instance of a *definition schema*. Given some SQL-data object, such as a view, a constraint, a domain, or a base table, the descriptor of that object, and of all the objects that it directly or indirectly references, are in the same cluster of catalogs. For example, no referential constraint and no <query expression> can reference any SQL-data object whose descriptors are not in the same cluster (that is, they cannot “cross” a cluster boundary).

Whether or not any catalog can occur simultaneously in more than one cluster is implementation-defined.

Within a cluster, no two catalogs have the same name.

4.27 SQL-data

SQL-data is any data described by schemas that is under the control of an SQL-implementation in an SQL-environment.

4.28 SQL-environment

An *SQL-environment* comprises the following:

— a conforming SQL-implementation see Clause 20, "Conformance";

— zero or more catalogs;

— zero or more <authorization identifier>s;

— zero or more <module>s; and

— the SQL-data described by the schemas in the catalogs.

An SQL-environment may have other implementation-defined contents.

The rules determining which <module>s are considered to be within an SQL-environment are implementation-defined.

4.29 Modules

A <module> is an object specified in the module language. A <module> is either an *SQL-client <module>* or an *SQL-session <module>*.

SQL-client <module>s are created and destroyed by implementation-defined mechanisms (which can include the granting and revoking of module privileges). SQL-client <module>s exist in the SQL-environment containing an SQL-client. The <routine>s of an SQL-client <module> are invoked by host language programs. The <language clause> of an SQL-client <module> specifies a host programming language.

SQL-session <module>s are implicitly-created <module>s for prepared SQL-statements (see Subclause 4.41, "SQL-sessions"). A <module> consists of:

— a <module name>,

— a <language clause>,

— a <module authorization clause> with either or both of a <module authorization identifier> and a <schema name>,

— an optional default SQL-path used to qualify:

- unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in the <module>,
- unqualified <abstract data type name>s that are immediately contained in a <user-defined type>s that are contained in the <module>,
- unqualified <distinct type name>s that are immediately contained in <user-defined type>s that are contained in the <module>, and
- unqualified <type template name>s that are contained in <user-defined type>s that are contained in the <module>,

— an optional <module character set specification> that identifies the character repertoire used for expressing the names of schema objects used in the <module>,

— zero or more <temporary table declaration>s,

— zero or more <temporary view declaration>s,

— zero or more <temporary abstract data type declaration>s,

— zero or more cursors specified by <declare cursor>s, and

— one or more <routine>s.

All <identifier>s contained in the <module> are expressed in either <SQL language character> or the character repertoire indicated by <module character set specification> unless they are specified with "<introducer>".

A compilation unit is a segment of executable code, possibly consisting of one or more subprograms. A <module> is associated with a compilation unit during its execution. A single <module> may be associated with multiple compilation units and multiple <module>s may be associated with a single compilation unit. The manner in which this association is specified, including the possible requirement for execution of some implementation-defined statement, is implementation-defined. Whether a compilation unit may invoke or transfer control to other compilation units, written in the same or a different programming language, is implementation-defined.

A <module> is described by a module descriptor. A module descriptor contains:

- the name of the <module>,
- the descriptor of the character set used for representing the <module>'s <identifier>s and <character string literal>s,
- the schema name used for implicit qualification of unqualified names in the <module>
- the <module authorization identifier>,
- the default SQL-path used in the <module> for implicit qualification of <routine name>s contained in <routine invocation>s and <abstract data type name>s, <distinct type name>s, and <type template name>s contained in <user-defined type>s.
- the <language clause> of the <module>.

4.30 Routines

****Editor's Note****

Paper X3H2-94-102/SOU-063, in item 3.9, contained the instruction "Leave an explanation of routines in Part 2 and a note to see Part 4's definition."

That instruction is not an adequate change proposal and the Editor requires that the authors of the paper, or other interested parties, provide the requisite text here.

4.31 SQL-paths

An SQL-path is a list of one or more <schema name>s used to search for one of the following:

— the subject routine of a <routine invocation> whose <routine name> does not contain a <local or schema qualifier>,

— the subject abstract data type when the <abstract data type name> does not contain a <schema name>,

— the subject distinct type when the <distinct type name> does not contain a <schema name>, and

— the subject type template when the <type template name> does not contain a <schema name>,

It is used only when SQL-invoked routines are being invoked or <user-defined type>s are being identified.

4.32 Parameters

A parameter is declared in a <routine> by a <parameter declaration>. A parameter either assumes or supplies the value of the corresponding argument in the call of the <routine>.

A parameter is either an SQL-supplied parameter or an externally-supplied parameter, depending on whether it is declared in an SQL-invoked or externally-invoked <routine>.

A <parameter declaration> specifies the <data type> of its value. The <data type>s of externally supplied parameters map to the host language types of the arguments.

SQL supplied parameters can be null. Externally supplied parameters cannot be null, except through the use of indicator parameters.

4.32.1 Status parameters

The SQLSTATE and SQLCODE parameters are status parameters. They are set to status codes that indicate either that a call of the <routine> completed successfully or that an exception condition was raised during execution of the <routine>.

Note: The SQLSTATE parameter is the preferred status parameter. The SQLCODE parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

A <routine> must specify either the SQLSTATE parameter or the SQLCODE parameter or both. The SQLSTATE parameter is a character string parameter for which exception values are defined in Clause 19, "Status codes". The SQLCODE parameter is an integer parameter for which the negative exception values are implementation-defined.

If a condition is raised that causes a statement to have no effect other than that associated with raising the condition (that is, not a completion condition), then the condition is said to be an *exception condition* or *exception*. If a condition is raised that permits a statement to have an effect other than that associated with raising the condition (corresponding to an SQLSTATE class value of *successful completion*, *warning*, or *no data*), then the condition is said to be a *completion condition*.

Exception conditions or completion conditions may be raised during the execution of an <SQL procedure statement>. One of the exception conditions becomes the active condition when the <SQL procedure statement> terminates. If the active condition is an exception condition, then it will be called the active exception condition. If the active condition is a completion condition, then it will be called the active completion condition.

The completion condition *warning* is broadly defined as completion in which the effects are correct, but there is reason to caution the user about those effects. It is raised for implementation-defined conditions as well as conditions specified in this
ANSI American
ISO International
Standard. The completion condition *no data* has special significance and is used to indicate an empty result. The completion condition *successful completion* is defined to indicate a completion condition that does not correspond to *warning* or *no data*. This includes conditions in which the SQLSTATE subclass provides implementation-defined information of a non-cautionary nature.

For the purpose of choosing status parameter values to be returned, exception conditions for transaction rollback have precedence over exception conditions for statement failure. Similarly, the completion condition *no data* has precedence over the completion condition *warning*, which has precedence over the completion condition *successful completion*. All exception conditions have precedence over all completion conditions. The values assigned to SQLSTATE shall obey these precedence requirements.

4.32.2 Data parameters

A data parameter is a parameter that is used to either assume or supply the value of data exchanged between a host program and an SQL-implementation.

4.32.3 Indicator parameters

An indicator parameter is an integer parameter that is specified immediately following another parameter. Its primary use is to indicate whether the value that the other parameter assumes or supplies is a null value and the appropriate null state of that null value, if any. An indicator parameter cannot immediately follow another indicator parameter.

The other use for indicator parameters is to indicate whether string data truncation occurred during a transfer between a host program and an SQL-implementation in parameters or host variables. If a non-null character string value is transferred and the length of the target data item is sufficient to accept the entire source data item, then the indicator parameter or variable is set to 0 to indicate that truncation did not occur. However, if the length of the target data item is insufficient, the indicator parameter or variable is set to the length (in characters or bits, as appropriate) of the source data item to indicate that truncation occurred and to indicate original length in characters or bits, as appropriate, of the source.

4.33 Diagnostics area

The diagnostics area is a place where completion and exception condition information is stored when an SQL-statement is executed. There is one diagnostics area associated with an SQL-agent, regardless of the number of <module>s that the SQL-agent includes or the number of connections in use.

At the beginning of the execution of any <routine> that contains an <SQL procedure statement> that is not an <SQL diagnostics statement>, the diagnostics area is emptied. An implementation must place information about a completion condition or an exception condition reported by SQLCODE or SQLSTATE into this area. If other conditions are raised, an implementation may place information about them into this area.

<routine>s containing <SQL diagnostics statement>s return a code indicating completion or exception conditions for that statement via SQLCODE or SQLSTATE, but do not modify the diagnostics area.

An SQL-agent may choose the size of the diagnostics area with the <set transaction statement>; if an SQL-agent does not specify the size of the diagnostics area, then the size of the diagnostics area is implementation-dependent, but shall always be able to hold information about at least one condition. An implementation may place information into this area about fewer conditions than are specified. The ordering of the information about conditions placed into the diagnostics area is implementation-dependent, except that the first condition in the diagnostics area always corresponds to the condition specified by the SQLSTATE or SQLCODE value.

4.34 Standard programming languages

This
ANSI American
ISO International
Standard specifies the actions of <routine>s in <module>s when those <routine>s are called by programs that conform to certain specified programming language standards. The term “standard *PLN* program”, where *PLN* is the name of a programming language, refers to a program that conforms to the standard for that programming language as specified in Clause 2, "Normative references".

Note: In this
ANSI American
ISO International
Standard, for the purposes of interfacing with programming languages, the data types DATE, TIME, TIMESTAMP, and INTERVAL must be converted to or from character strings in those programming languages by means of a <cast specification>. It is anticipated that future evolution of programming language standards will support data types corresponding to these four SQL data types; this standard will then be amended to reflect the availability of those corresponding data types. The data type CHARACTER is also mapped to character strings in the programming languages. However, because the facilities available in the programming languages do not provide the same capabilities as those available in SQL, there must be agreement between the host program and SQL regarding the specific format of the character data being exchanged. Specific syntax for this agreement is provided in thie
ANSI American
ISO International
standard. For standard programming languages, C, COBOL, Fortran, and Pascal, bit strings are mapped to character variables in the host language in a manner described in Subclause 14.1, "<embedded SQL host program>" (in Part 5 of this
ANSI American
ISO International
Standard). For standard programming languages Ada and PL/I, bit string variables are directly supported.

****Editor’s Note****

The preceding Note points to a Subclause in a different document (SQL/Bindings), which we explicitly want to avoid. How can this be resolved?

****Editor's Note****
Standard programming language MUMPS is not mentioned herein. It should be resolved in some way. Do we want to define handling of BITs in MUMPS?

4.35 Cursors

A cursor is specified by a <declare cursor>.

For every <declare cursor> in a <module>, a cursor is effectively created when an SQL-transaction (see Subclause 4.39, "SQL-transactions") referencing the <module> is initiated.

One of the properties that may be specified for a cursor determines whether or not it is a *holdable-cursor*:

— A cursor that is not a holdable-cursor is closed when the SQL-transaction in which it was created is terminated.

— A holdable-cursor is not closed if that cursor is in the open state at the time that the SQL-transaction is terminated with a commit operation. A holdable-cursor that is in the closed state at the time that the SQL-transaction is terminated remains closed. A holdable-cursor is closed no matter what its state if the SQL-transaction is terminated with a rollback operation.

— A holdable-cursor is destroyed when the SQL-session in which it was created is terminated.

Note: A holdable-cursor may be said to be "holdable" or "held".

A cursor is in either the open state or the closed state. The initial state of a cursor is the closed state. A cursor is placed in the open state by an <open statement> and returned to the closed state by a <close statement> or a <rollback statement>. An open cursor that was not defined as a holdable-cursor is also closed by a <commit statement>.

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

When the ordering of a cursor is not defined by an <order by clause>, the relative position of two rows is implementation-dependent. When the ordering of a cursor is partially determined by an <order by clause>, then the relative positions of two rows are determined only by the <order by clause>; if the two rows have equal values for the purpose of evaluating the <order by clause>, then their relative positions are implementation-dependent.

A cursor is either *inherently updatable* or *not inherently updatable*. If the table identified by a cursor is not inherently updatable or if INSENSITIVE is specified for the cursor, then the cursor is *not inherently updatable*; otherwise, the cursor is inherently updatable. The operations of insert, update, and delete are permitted for inherently updatable cusors, subject to constraining Access Rules. These operations may also be permitted for cursors that are not inherently updatable if appropriate triggers have been defined.

The position of a cursor in the open state is either before a certain row, on a certain row, or after the last row. If a cursor is on a row, then that row is the current row of the cursor. A cursor may be before the first row or after the last row of a table even though the table is empty. When a cursor is initially opened, the position of the cursor is before the first row.

A holdable-cursor that has been held open retains its position when the new SQL-transaction is initiated. However, before either an <update statement: positioned> or a <delete statement: positioned> is permitted to reference that cursor in the new SQL-transaction, a <fetch statement> must be issued against the cursor.

A <fetch statement> positions an open cursor on a specified row of the cursor's ordering and retrieves the values of the columns of that row. An <update statement: positioned> updates the current row of the cursor. A <delete statement: positioned> deletes the current row of the cursor.

If an error occurs during the execution of an SQL-statement that identifies a cursor, then, except where otherwise explicitly defined, the effect, if any, on the position or state of that cursor is implementation-dependent.

If a completion condition is raised during the execution of an SQL-statement that identifies a cursor, then the particular SQL-statement identifying that open cursor on which the completion condition is returned is implementation-dependent.

The following paragraphs define several terms used to discuss issues relating to cursor sensitivity:

A *DB-change* is any change to SQL-data resulting from the execution of an <insert statement>, an <update statement: positioned>, an <update statement: searched>, a <delete statement: positioned> or a <delete statement: searched>.

A DB-change is said to be *independent* of a cursor *CR* if and only if it is not made by an <update statement: positioned> or a <delete statement: positioned> that is positioned on *CR*.

A DB-change is said to be *significant* to *CR* if and only if it is independent of *CR*, and, had it occurred before *CR* was opened, would have caused the table associated with the cursor to be different in any respect.

If *CR* denotes an open cursor, then let **proc-VC** be the procedure:

— Let *P* be the position of *CR* (either before the first row, after the last row, or on or before some row *R* of the table associated with *CR*).

— *CR* is closed.

— The DB-change is made, including any relevant referential actions.

— *CR* is reopened by applying the General Rules of Subclause 13.2, "<open statement>", using the same *S* that resulting from applying those General Rules in the course of executing the most recent <open statement> for *CR*.

— If the rows that constitute *CR* or the order of those rows has changed as a result of closing and reopening *CR*, then the position of *CR* is undefined. Otherwise, the position of *CR* is set to *P*.

A DB-change is said to be *invisible* to *CR* if and only if it has no effect on *CR*.

A DB-change is said to be *visible* to *CR* if and only if its effect on *CR* is as if the change had been effected by executing the procedure **proc-VC** on *CR*.

If a cursor is open, and the current SQL-transaction makes a change to SQL-data other than through that cursor, and the <declare cursor> of that cursor did not specify either SENSITIVE or INSENSITIVE, then whether any effect of that change will be visible through that cursor before it is closed is implementation-dependent.

If a holdable-cursor that is not an INSENSITIVE cursor is held open for a subsequent SQL-transaction, then whether the effects of any changes made to SQL-data other than through that cursor (by this or any other SQL-transaction) will be visible through that cursor in the subsequent SQL-transaction before that cursor is closed is implementation-defined.

If a cursor is open, and the current SQL-transaction makes a change to SQL-data other than through that cursor and the <declare cursor> for that cursor specified INSENSITIVE, then the effect of that change will not be visible through that cursor before it is closed. Otherwise, whether the effect of such a change will be visible through that cursor before it is closed is implementation-dependent.

4.36 SQL-statements

****Editor's Note****
This Subclause has not been properly updated for the various SQL3 statements and SQL3 work that has been done. See Possible Problem **261**.

4.36.1 Classes of SQL-statements

An SQL-statement is a string of characters that conforms to the Format and Syntax Rules specified in the Parts of this
ANSI American
ISO International
Standard. Most SQL-statements can be prepared for execution and executed in a module, in which case they are prepared when the module is created and executed when the containing procedure is called (see Subclause 4.29, "Modules").

There are at least two ways of classifying SQL-statements:

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

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

This
ANSI American
ISO International
Standard permits implementations to provide additional, implementation-defined, statements that may fall into any of these categories. This Subclause will not mention those statements again, as their classification is entirely implementation-defined.

4.36.2 SQL-statements classified by function

The following are the main classes of SQL-statements:

— SQL-schema statements; these may have a persistent effect on the set of schemas

— SQL-data statements; some of these, the SQL-data change statements, may have a persistent effect on SQL-data

— SQL-transaction statements; except for the <commit statement>, these, and the following classes, have no effects that persist when a session is terminated

— SQL-control statements

— SQL-connection statements

— SQL-session statements

— SQL-diagnostics statements

The following are the SQL-schema statements:

— <schema definition>

— <alter schema statement>

— <drop schema statement>

— <domain definition>

— <drop domain statement>

— <table definition>

— <drop table statement>

— <view definition>

— <drop view statement>

— <assertion definition>

— <drop assertion statement>

— <alter table statement>

— <alter domain statement>

— <grant statement>

— <revoke statement>

— <character set definition>

— <drop character set statement>

— <collation definition>

— <drop collation statement>

— <translation definition>

— <drop translation statement>

— <trigger definition>

— <drop trigger statement>

— <null class definition>

— <drop null class statement>

• 2 list elements moved to Part 4

— <abstract data type definition>

— <type template definition>

— <drop type template statement>

— <drop data type statement>

— <distinct type definition>

— <operations definition>

ANSI Only–SQL3

— <default role definition>

— <drop default role statement>

— <role definition>

— <grant role statement>

— <revoke role statement>

— <drop role statement>

— <module>

The following are the SQL-data statements:

— <temporary table declaration>

— <temporary abstract data type declaration>

— <temporary view declaration>

— <declare cursor>

— <open statement>

— <close statement>

— <fetch statement>

— <select statement: single row>

— <free locator statement>

• 2 statements deleted.

— All SQL-data change statements

The following are the SQL-data change statements:

— <insert statement>

— <delete statement: searched>

— <delete statement: positioned>

— <update statement: searched>

— <update statement: positioned>

The following are the SQL-transaction statements:

— <start transaction statement>

— <set transaction statement>

— <set constraints mode statement>

— <commit statement>

— <rollback statement>

— <test completion statement>

— <savepoint statement>

— <release savepoint statement>

The following are the SQL-connection statements:

— <connect statement>

— <set connection statement>

— <disconnect statement>

The following are the SQL-session statements:

— <set session characteristics statement>

— <set session authorization identifier statement>

— <set role statement>

— <set local time zone statement>

The following are the SQL-diagnostics statements:

— <get diagnostics statement>

4.36.3 SQL-statements and transaction states

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

— All SQL-schema statements

— The SQL-transaction statements <commit statement> and <rollback statement>, if they specify AND CHAIN.

— 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>
- <free locator statement>

— <start transaction statement>

The following SQL-statements are not transaction-initiating SQL-statements, *i.e.*, if there is no current SQL-transaction, and a statement of this class is executed, no SQL-transaction is initiated.

— All SQL-transaction statements except <start transaction statement>s and <commit statement>s and <rollback statement>s that specify AND CHAIN.

— All SQL-connection statements

— All SQL-session statements

— All SQL-diagnostics statements

— The following SQL-data statements:

- <temporary table declaration>
- <temporary abstract data type declaration>
- <temporary view declaration>
- <declare cursor>

If the initiation of an SQL-transaction occurs in an atomic execution context, and an SQL-transaction has already completed in this context, then an exception condition is raised: *invalid transaction initiation*.

4.36.4 Asynchronous execution

An <SQL procedure statement> may be executed either synchronously or asynchronously. If an <SQL procedure statement> is executed synchronously, then control is not returned to the SQL-agent until all actions specified for the statement have been performed. If an <SQL procedure statement> is executed asynchronously, then, at the option of the DBMS, control may be returned to the SQL-agent before any of the actions specified for the statement have been performed. Synchronous and asynchronous execution of <SQL procedure statement>s may be intermixed, but the effect of the <SQL procedure statement>s is the same as if they had all been executed synchronously in the order in which they had been initiated. Furthermore, an <SQL procedure statement> initiated for asynchronous execution may be either accepted or rejected by the DBMS. In the latter case, an exception condition is raised: *asynchronous SQL statement not accepted*.

4.36.5 SQL-statement atomicity

The execution of all SQL-statements is atomic. The evaluation of a <subquery> is atomic.

An *atomic execution context* is said to be active during the execution of an atomic SQL-statement or evaluation of a <subquery> that is atomic. Within one atomic execution context, another atomic execution context may become active. This latter atomic execution context is said to be a *more recent atomic execution context*. During the execution of any SQL-statement *S*, if there is an atomic execution context for which no other atomic execution context is more recent, then it is the *most recent atomic execution context*.

4.37 Privileges and roles

A privilege authorizes a given category of <action> to be performed on a specified base table, view, column, domain, character set, collation, translation, abstract data type, type template, external function, null class, trigger, or module or authorizes EXTERNAL PRIVILEGES on specified external functions by a specified <authorization identifier>. The mapping of <authorization identifier>s to operating system users is implementation-dependent. The <action>s that can be specified are:

— INSERT

— INSERT (<column name list>)

— UPDATE

— UPDATE (<column name list>)

— DELETE

— SELECT

— SELECT (<column name list>)

— REFERENCES

— REFERENCES (<column name list>)

— USAGE

— UNDER

— TRIGGER

— EXECUTE

A role, identified by a <role name>, is a set of privileges defined by the privilege descriptors whose grantee is either that role name or the role name of one of the set of roles defined by the role authorization descriptors whose grantee is the first role name. A role may be granted to either <authorization identifier>s or to other roles with a <grant role statement>. The WITH
ANSI GRANT
ISO ADMIN
OPTION clause of the <grant role statement> specifies whether the recipient of a role may grant it to others. Each grant is represented and identified by a *role authorization descriptor*. A role authorization descriptor identifies the role, grantee, and whether the role was granted with the WITH
ANSI GRANT
ISO ADMIN
OPTION and hence is grantable.

An <authorization identifier> is specified for each schema and <module> as well as for each SQL-session.

A schema that is owned by a given <authorization identifier> may contain privilege descriptors that describe privileges granted to other <authorization identifier>s or to roles (grantees). The granted privileges apply to objects defined in the current schema. The WITH GRANT OPTION clause of a <grant statement> specifies whether the <authorization identifier> recipient of a privilege (acting as a grantor) may grant it to others.

When an SQL-session is initiated, the <authorization identifier> for the SQL-session, called the *SQL-session <authorization identifier>*, is determined in an implementation-dependent manner, unless the session is initiated using a <connect statement>. Subsequently, the SQL-session <authorization identifier> can be redefined by the successful execution of a <set session authorization identifier statement>.

A <module> may specify an <authorization identifier>, called a <module authorization identifier>. If the <module authorization identifier> is specified, then that <module authorization identifier> is used as the *current <authorization identifier>* for the execution of all <routine>s in the <module>. If the <module authorization identifier> is not specified, then the SQL-session <authorization identifier> is used as the current <authorization identifier> for the execution of each <routine> in the <module>.

A <schema definition> may specify an <authorization identifier>, called a <schema authorization identifier>. If the <schema authorization identifier> is specified, then that is used as the *current <authorization identifier>* for the creation of the schema. If the <schema authorization identifier> is not specified, then the <module authorization identifier> or the SQL-session <authorization identifier> is used as the current <authorization identifier> for the creation of the schema.

The current <authorization identifier> determines the privileges for the execution of each SQL-statement.

Each privilege is represented by a privilege descriptor. A privilege descriptor contains:

— the identification of the table, column, domain, character set, collation, translation, abstract data type, type template, external function, null class, trigger, or module that the descriptor describes;

— the <authorization identifier> of the grantor of the privilege;

— the <authorization identifier> of the grantee of the privilege;

— identification of the action that the privilege allows; and

— an indication of whether or not the privilege is grantable.

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 identified by the 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 identified by the privilege descriptor.

A table privilege descriptor specifies that the privilege identified by the action (unless the action is DELETE) is to be automatically granted by the grantor to the grantee on all columns subsequently added to the table.

A privilege descriptor with an action of USAGE is called a usage privilege descriptor and identifies the existence of a privilege on the domain, abstract data type, type template, null class, character set, collation, or translation identified by the privilege descriptor.

A privilege descriptor with an action of UNDER is called an under privilege descriptor and identifies the existence of the privilege on the abstract data type identified by the privilege descriptor. It authorizes the grantee to create one object UNDER another in a hierarchy of types.

A privilege descriptor with an action of EXECUTE is called an execute privilege descriptor and identifies the existence of a privilege on the <module> identified by the privilege descriptor.

A privilege descriptor that specifies EXTERNAL PRIVILEGES is called an external call privilege descriptor and identifies the existence of a privilege on the external function identified by the privilege descriptor.

Because roles may be granted to other roles, a role is said to "contain" other roles. The set of roles *X* contained in any role A is defined as the set of roles identified by role authorization descriptors whose grantee is *A*, together with all other roles contained by roles in *X*.

A grantable privilege is a privilege associated with a schema that may be granted by a <grant statement>.

The phrase *applicable roles* refers to the roles defined by role authorization descriptors as having been granted to the SQL-session <authorization identifier> or to PUBLIC together with all other roles they contain.

ANSI Only–SQL3

Enabled applicable roles refers to the roles enabled by a <set role statement>. The phrase *default role* refers to the role that becomes enabled at the time of SQL-session initialization and is create by <create default role> statement. If a default role is created prior to SQL-session initialization, it is enabled; otherwise no roles are enabled. The execution of a <set role statement> disables any roles that may have been enabled by any previous <set role statement> and then enables the role specified in the statement.

A default role is described by a default role descriptor. Every default role descriptor includes:

— the name of the default role;

— the <authorization identifier> of the creator of the default role;

ISO Only—caused by ANSI changes not yet considered by ISO

Enabled applicable roles refers to the roles enabled by a <set role statement>. When an SQL-session is initiated, no roles are enabled. The execution of a <set role statement> disables any roles that may have been enabled by any previous <set role statement> and then enables the role specified in the statement, together with all roles contained in that role, for that user session.

The phrase *user privileges* refers to the set of privileges defined by the privilege descriptors whose grantee is either the identified <authorization identifier> or PUBLIC.

The phrase *applicable privileges* of enabled applicable roles refers to the set of privileges defined by the privilege descriptors whose grantee is one of the enabled applicable roles.

The phrase *applicable privileges* refers to the set of privileges defined by user privileges of the current <authorization identifier> together with those defined by applicable privileges of enabled applicable roles.

Privilege descriptors that represent privileges for the owner of an object have a special grantor value, "_SYSTEM". This value is reflected in the Information Schema for all privileges that apply to the owner of the object.

4.38 SQL-agents

An SQL-agent is an implementation-dependent entity that causes the execution of SQL-statements.

4.39 SQL-transactions

An *SQL-transaction* (transaction) is a sequence of executions of SQL-statements that is atomic with respect to recovery. These operations are performed by one or more compilation units and <module>s.

It is implementation-defined whether or not the execution of an SQL-data statement is permitted to occur within the same SQL-transaction as the execution of an SQL-schema statement. If it does occur, then the effect on any open cursor or deferred constraint is implementation-defined. There may be additional implementation-defined restrictions, requirements, and conditions. If any such restrictions, requirements, or conditions are violated, then an implementation-defined exception condition or a completion condition *warning* with an implementation-defined subclass code is raised.

Each <module> that executes an SQL-statement of an SQL-transaction is associated with that SQL-transaction. An SQL-transaction is initiated when no SQL-transaction is currently active and a <routine> is called that results in the execution of a *transaction-initiating* SQL-statement. An SQL-transaction is terminated by a <commit statement> or a <rollback statement>. If an SQL-transaction is terminated by successful execution of a <commit statement>, then all changes made to SQL-data or schemas by that SQL-transaction are made persistent and accessible to all concurrent and subsequent SQL-transactions. If an SQL-transaction is terminated by a <rollback

statement> or unsuccessful execution of a <commit statement>, then all changes made to SQL-data or schemas by that SQL-transaction are canceled. Committed changes cannot be canceled. If execution of a <commit statement> is attempted, but certain exception conditions are raised, it is unknown whether or not the changes made to SQL-data or schemas by that SQL-transaction are canceled or made persistent.

An SQL-transaction may be partially rolled back by using a savepoint. The savepoint and its <savepoint name> are established within an SQL-transaction when a <savepoint statement> is executed. If a <rollback statement> references a savepoint, then all changes made to SQL-data or schema subsequent to the establishment of the savepoint are canceled, and the SQL-transaction is restored to its state as it was immediately following the execution of the <savepoint statement>. Savepoints are destroyed when an SQL-transaction is terminated, or when a <release savepoint statement> is executed. Savepoints may be redefined within an SQL-transaction by executing a <savepoint statement> which refers to a savepoint previously defined in the same SQL-transaction.

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

An SQL-transaction has a *constraint mode* for each integrity constraint. The constraint mode for an integrity constraint in an SQL-transaction is described in Subclause 4.21, "Integrity constraints".

An SQL-transaction has an access mode that is either *read-only* or *read-write*. The access mode may be explicitly set by a <set transaction statement>; otherwise, it is implicitly set to *read-write*. The term *read-only* applies only to viewed tables and persistent base tables.

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

An SQL-transaction has a cursor mode that is either *cascade on* or *cascade off*. The cursor mode is initially *cascade on*. The cursor mode may be explicitly set by an option on the <open statement> that opens a cursor. Opening a cursor CASCADE OFF causes the checking of all table constraints and assertions affected by that cursor to be effectively deferred, the execution of all <referential action>s, pendant actions, and <triggered SQL statement>s affected by that cursor to be effectively deferred, and the fixing of all matching rows and unique matching rows. Closing the cursor causes an effective check of all table constraints and assertions, the effective execution of all referential actions, pendant actions, triggered SQL statements, and the re-evaluation of all matching rows and unique matching rows.

When the cursor mode of an SQL-transaction is *cascade off*, the SQL-transaction is said to be in *set-processing mode*. The interval between opening a cursor CASCADE OFF and closing that cursor is called the *set-processing mode session* for the SQL-transaction.

SQL-transactions initiated by different SQL-agents that access the same SQL-data or schemas and overlap in time are *concurrent SQL-transactions*.

An SQL-transaction has an *isolation level* that is READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE. The isolation level of an SQL-transaction defines the degree to which the operations on SQL-data or schemas in that SQL-transaction are affected by the effects of and can affect operations on SQL-data or schemas in concurrent SQL-transactions. The isolation level of an SQL-transaction when any cursor is held open from the previous SQL-transaction within an SQL-session is the isolation level of the previous SQL-transaction by default. If no cursor is held open, or this is the first SQL-transaction within an SQL-session, then the isolation level is SERIALIZABLE by default. The level can be explicitly set by the <set transaction statement>.

The execution of concurrent SQL-transactions at isolation level SERIALIZABLE is guaranteed to be serializable. A serializable execution is defined to be an execution of the operations of concurrently executing SQL-transactions that produces the same effect as some serial execution of those same SQL-transactions. A serial execution is one in which each SQL-transaction executes to completion before the next SQL-transaction begins.

The isolation level specifies the kind of phenomena that can occur during the execution of concurrent SQL-transactions. The following phenomena are possible:

1) *P1* (“Dirty read”): SQL-transaction *T1* modifies a row. SQL-transaction *T2* then reads that row before *T1* performs a COMMIT. If *T1* then performs a ROLLBACK, *T2* will have read a row that was never committed and that may thus be considered to have never existed.

2) *P2* (“Non-repeatable read”): SQL-transaction *T1* reads a row. SQL-transaction *T2* then modifies or deletes that row and performs a COMMIT. If *T1* then attempts to reread the row, it may receive the modified value or discover that the row has been deleted.

3) *P3* (“Phantom”): SQL-transaction *T1* reads the set of rows *N* that satisfy some <search condition>. SQL-transaction *T2* then executes SQL-statements that generate one or more rows that satisfy the <search condition> used by SQL-transaction *T1*. If SQL-transaction *T1* then repeats the initial read with the same <search condition>, it obtains a different collection of rows.

The four isolation levels guarantee that each SQL-transaction will be executed completely or not at all, and that no updates will be lost. The isolation levels are different with respect to phenomena *P1*, *P2*, and *P3*. Table 10, "SQL-transaction isolation levels and the three phenomena" specifies the phenomena that are possible and not possible for a given isolation level.

Table 10—SQL-transaction isolation levels and the three phenomena

Level	***P1***	***P2***	***P3***
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Not Possible	Possible	Possible
REPEATABLE READ	Not Possible	Not Possible	Possible
SERIALIZABLE	Not Possible	Not Possible	Not Possible

Note: The exclusion of these phenomena for SQL-transactions executing at isolation level SERIALIZABLE is a consequence of the requirement that such transactions be serializable.

Changes made to SQL-data or schemas by an SQL-transaction in an SQL-session may be perceived by that SQL-transaction in that same SQL-session, and by other SQL-transactions, or by that same SQL-transaction in other SQL-sessions, at isolation level READ UNCOMMITTED, but cannot be perceived by other SQL-transactions at isolation level READ COMMITTED, REPEATABLE READ, or SERIALIZABLE until the former SQL-transaction terminates with a <commit statement>.

Regardless of the isolation level of the SQL-transaction, phenomena *P1*, *P2*, and *P3* shall not occur during the implied reading of schema definitions performed on behalf of executing an SQL-statement, the checking of integrity constraints, and the execution of referential actions associated with referential constraints. The schema definitions that are implicitly read are implementation-dependent. This does not affect the explicit reading of rows from tables in the Information Schema, which is done at the isolation level of the SQL-transaction.

The execution of a <rollback statement> may be initiated implicitly by an implementation when it detects the inability to guarantee the serializability of two or more concurrent SQL-transactions. When this error occurs, an exception condition is raised: *transaction rollback—serialization failure*.

The execution of a <rollback statement> may be initiated implicitly by an implementation when it detects unrecoverable errors. When such an error occurs, an exception condition is raised: *transaction rollback* with an implementation-defined subclass code.

The execution of an SQL-statement within an SQL-transaction has no effect on SQL-data or schemas other than the effect stated in the General Rules for that SQL-statement, in the General Rules for Subclause 11.12, "<referential constraint definition>", in the General Rules for Subclause 11.45, "<trigger definition>", and in the General Rules for Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard. Together with serializable execution, this implies that all read operations are repeatable within an SQL-transaction at isolation level SERIALIZABLE, except for:

1) the effects of changes to SQL-data or schemas and its contents made explicitly by the SQL-transaction itself,

2) the effects of differences in parameter values supplied to procedures, and

3) the effects of references to time-varying system variables such as CURRENT_DATE and CURRENT_USER.

In some environments (*e.g.*, remote database access), an SQL-transaction can be part of an encompassing transaction that is controlled by an agent other than the SQL-agent. The encompassing transaction may involve different resource managers, the SQL-implementation being just one instance of such a manager. In such environments, an encompassing transaction must be terminated via that other agent, which in turn interacts with the SQL-implementation via an interface that may be different from SQL (COMMIT or ROLLBACK), in order to coördinate the orderly termination of the encompassing transaction. When an SQL-transaction is part of an encompassing transaction that is controlled by an agent other than an SQL-agent and a <rollback statement> is initiated implicitly by an implementation, then the implementation will interact with that other agent to terminate that encompassing transaction. The specification of the interface between such agents and the SQL-implementation is beyond the scope of this
ANSI American
ISO International
standard. However, it is important to note that the semantics of an SQL-transactaion remain as defined in the following sense:

— When an agent that is different from the SQL-agent requests the SQL-implementation to rollback an SQL-transaction, the General Rules of Subclause 14.8, "<rollback statement>", are performed.

— When such an agent requests the SQL-implementation to commit an SQL-transaction, the General Rules of Subclause 14.7, "<commit statement>", are performed. To guarantee orderly termination of the encompassing transaction, this commit operation may be processed in several phases not visible to the application; not all the General Rules of Subclause 14.7, "<commit statement>", need to be executed in a single phase.

However, even in such environments, the SQL-agent interacts directly with the SQL-server to set attributes (such as *read-only* or *read-write*, isolation level, and constraints mode) that are specific to the SQL-transaction model.

It is implementation-defined whether SQL-transactions that affect more than one SQL-server are supported. If such SQL-transactions are supported, then they generally have all the same characteristics (access mode, diagnostics area limit, and isolation level, as well as constraint mode and cursor mode). However, it is possible to alter some characteristics of such an SQL-transaction at one SQL-server by the use of the SET LOCAL TRANSACTION statement; if a SET LOCAL TRANSACTION statement is executed at an SQL-server before any transaction-initiating SQL-statement, then it may set the characteristics of that *branch* of the SQL-transaction at that SQL-server.

The characteristics of a branch of an SQL-transaction are limited by the characteristics of the SQL transaction as a whole:

— If the SQL-transaction is read-write, then the branch of the SQL-transaction may be read-write or read-only; if the SQL-transaction is read-only, then the branch of the SQL-transaction must be read-only.

— If the SQL-transaction has an isolation level of READ UNCOMMITTED, then the branch of the SQL-transaction may have an isolation level of READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, or SERIALIZABLE.

  If the SQL-transaction has an isolation level of READ COMMITTED, then the branch of the SQL-transaction must have an isolation level of READ COMMITTED, REPEATABLE READ, or SERIALIZABLE.

  If the SQL-transaction has an isolation level of REPEATABLE READ, then the branch of the SQL-transaction must have an isolation level of REPEATABLE READ or SERIALIZABLE.

  If the SQL-transaction has an isolation level of SERIALIZABLE, then the branch of the SQL-transaction must have an isolation level of SERIALIZABLE.

— The diagnostics area limit of a branch of an SQL-transaction is always the same as the diagnostics area limit of the SQL-transaction; SET LOCAL TRANSACTION shall not specify a diagnostics area limit.

SQL-transactions that are not part of an encompassing transaction are terminated by the execution of <commit statement>s and <rollback statement>s. If those statements specify AND CHAIN, then they also initiate a new SQL-transaction with the same characteristics as the SQL-transaction that was just terminated, except that the constraint mode of all integrity constraints revert to their default mode (*deferred* or *immediate*).

4.40 SQL-connections

An *SQL-connection* is an association between an SQL-client and an SQL-server. An SQL-connection may be established and named by a <connect statement>, which identifies the desired SQL-server by means of an <SQL-server name>. A <connection name> is specified as a <simple value specification> whose value is an <identifier>. Two <connection name>s identify the same SQL-connection if their values, with leading and trailing <space>s removed, are equivalent according to the rules for <identifier> comparison in Subclause 5.2, "<token> and <separator>". It is implementation-defined how an implementation uses <SQL-server name> to determine the location, identity, and communication protocol required to access the SQL-server and create an SQL-session.

An SQL-connection is an *active SQL-connection* if any SQL-statement that initiates or requires an SQL-transaction has been executed at its SQL-server via that SQL-connection during the current SQL-transaction.

An SQL-connection is either *current* or *dormant*. If the SQL-connection established by the most recently executed implicit or explicit <connect statement> or <set connection statement> has not been terminated, then that SQL-connection is the *current SQL-connection*; otherwise, there is no current SQL-connection. An existing SQL-connection that is not the current SQL-connection is a *dormant SQL-connection*.

An SQL implementation may detect the loss of the current SQL-connection during execution of any SQL-statement. When such a connection failure is detected, an exception condition is raised: *connection exception—statement completion unknown*. This exception condition indicates that the results of the actions performed in the SQL-server on behalf of the statement are unknown to the SQL-agent.

Similarly, an SQL-implementation may detect the loss of the current SQL-connection during the execution of a <commit statement>. When such a connection failure is detected, an exception condition is raised: *connection exception—transaction resolution unknown*. This exception condition indicates that the SQL-implementation cannot verify whether the SQL-transaction was committed successfully, rolled back, or left active.

A user may initiate an SQL-connection between the SQL-client associated with the SQL-agent and a specific SQL-server by executing a <connect statement>. Otherwise, an SQL-connection between the SQL-client and an implementation-defined default SQL-server is initiated when a <routine> is called and no SQL-connection is current. The SQL-connection associated with an implementation-defined default SQL-server is called the *default SQL-connection*. An SQL-connection is terminated either by executing a <disconnect statement> or following the last call to a <routine> within the last active <module>. The mechanism and rules by which an SQL-implementation determines whether a call to a <routine> is the last call within the last active <module> are implementation-defined.

An implementation must support at least one SQL-connection and may require that the SQL-server be identified at the binding time chosen by the implementation. If an implementation permits more than one concurrent SQL-connection, then the SQL-agent may connect to more than one SQL-server and select the SQL-server by executing a <set connection statement>.

4.41 SQL-sessions

An *SQL-session* spans the execution of a sequence of consecutive SQL-statements invoked by a single user from a single SQL-agent.

An SQL-session is associated with an SQL-connection. The SQL-session associated with the default SQL-connection is called the *default SQL-session*. An SQL-session is either *current* or *dormant*. The *current SQL-session* is the SQL-session associated with the current SQL-connection. A *dormant SQL-session* is an SQL-session that is associated with a dormant SQL-connection.

Within an SQL-session, module local temporary tables are effectively created by <temporary table declaration>s and temporary views by <temporary view declaration>s. Module local temporary tables are accessible only to invocations of <routine>s in the <module> in which they are created. The definitions of module local temporary tables and temporary views persist until the end of the SQL-session.

Within an SQL-session, locators are effectively created when a host variable of a binary large object locator type or a character large object locator type is the target of a Retrieval Assignment. These locators are part of the context of an SQL-session and persist until the end of the current SQL-transaction.

An SQL-session has a unique implementation-dependent SQL-session identifier. This SQL-session identifier is different from the SQL-session identifier of any other concurrent SQL-session. The SQL-session identifier is used to effectively define implementation-defined schemas that contain the instances of any global temporary tables, created local temporary tables, or declared local temporary tables within the SQL-session.

An SQL-session has an <authorization identifier> that is initially set to an implementation-defined value when the SQL-session is started, unless the SQL-session is started as a result of successful execution of a <connect statement>, in which case the initial <authorization identifier> of the SQL-session is set to the value of the implicit or explicit <user name> contained in the <connect statement>.

An SQL-session has a default local time zone displacement, which is a value of data type INTERVAL HOUR TO MINUTE. The default local time zone displacement is initially set to an implementation-defined value but can subsequently be changed by successful execution of a <set local time zone statement>.

An SQL-session has *enduring characteristics*. The enduring charactistics of an SQL-session are initially the same as the default values for the corresponding SQL-session attributes. The enduring characteristics are changed by successful execution of a <set session characteristics statement> that specifies one or more enduring characteristics. Enduring characteristics that are not specified in a <set session characteristics statement> are not changed in any way by the successful execution of that statement.

SQL-sessions have the following enduring characteristics:

— *enduring transaction characteristics*

Each of the enduring characteristics are affected by a corresponding alternative in the <session attribute> appearing in the <session attribute list> of a <set session characteristics statement>.

An SQL-session has context that is preserved when an SQL-session is made dormant and restored when the SQL-session is made current. This context comprises:

— the current SQL-session identifier,

— the current <authorization identifier>,

— the identities of all instances of temporary tables,

— the current default time zone,

— the current constraint mode for each integrity constraint,

— the current transaction access mode,

— the cursor position of all open cursors,

— the current transaction isolation level,

— the current transaction diagnostics area limit, and

— the value of all locators created within the current SQL-transaction

4.42 Client-server operation

As perceived by an SQL-agent, an SQL-implementation consists of one or more SQL-servers and one SQL-client through which SQL-connections can be made to the SQL-server or SQL-servers.

When an SQL-agent is active, it is bound in some implementation-defined manner to a single SQL-client. That SQL-client processes the explicit or implicit <SQL connection statement> for the first call to a <routine> by an SQL-agent. The SQL-client communicates with, either directly or possibly through other agents such as RDA, one or more SQL-servers. An SQL-session involves an SQL-agent, an SQL-client, and a single SQL-server.

SQL-client <module>s associated with the SQL-agent exist in the SQL-environment containing the SQL-client associated with the SQL-agent.

Called <routine>s containing an <SQL connection statement> or an <SQL diagnostics statement> are processed by the SQL-client. Following the successful execution of a <connect statement> or a <set connection statement>, the SQL-client <module>s associated with the SQL-agent are effectively materialized with an implementation-dependent <module name> in the SQL-server. Other called <routine>s are processed by the SQL-server.

A call by the SQL-agent to a <routine> containing an <SQL diagnostics statement> fetches information from the diagnostics area associated with the SQL-client. Following the execution of an <SQL procedure statement> by an SQL-server, diagnostic information is passed in an implementation-dependent manner into the SQL-agent's diagnostics area in the SQL-client. The effect on diagnostic information of incompatibilities between the character repertoires supported by the SQL-client and SQL-server is implementation-dependent.

4.43 Information Schema

In each catalog in an SQL-environment, there is a schema, the Information Schema, with the name INFORMATION_SCHEMA, containing a number of view descriptors, one base table descriptor, and several domain descriptors. The data accessible through these views is a representation of all of the descriptors in all of the schemas in that catalog. The <query expression> of each view ensures that a given user can access only those rows of the view that represent descriptors on which he has privileges. The rows of each view are required to represent correctly the descriptors in the catalog as they existed at the start of the current SQL-transaction, as modified subsequently by any changes made by the current SQL-transaction. The SELECT privilege is granted on each of the Information Schema views to PUBLIC WITH GRANT OPTION so they can be queried by any user and so that the SELECT privilege can be further granted on views that reference the Information Schema views. No further privilege is granted on them, so they cannot be updated.

The viewed tables in INFORMATION_SCHEMA are defined in terms of a collection of base tables in a schema named DEFINITION_SCHEMA, the Definition Schema. The only purpose of the definition of these base tables is to provide a data model to support the Information Schema. An implementation need do no more than simulate the existence of the base tables as viewed through the Information Schema views.

The Information Schema describes itself. It does not describe the base tables or views of the Definition Schema. If an implementation has defined additional objects that are associated with INFORMATION_SCHEMA, then those objects shall also be described in the Information Schema views.

4.44 Leveling

Four levels of conformance are specified in this
ANSI American
ISO International
Standard.

Entry SQL includes statements for defining schemas, data manipulation language, referential integrity, check constraints, and default clause from
ANSI ANSI X3.135-1992,
ISO ISO/IEC 9075:1992,
and options for module language. It also includes features related to deprecated features from
ANSI ANSI X3.135-1992
ISO ISO/IEC 9075:1992
(commas and parentheses in parameter lists, the SQLSTATE parameter, and renaming columns in the <select list>), features related to incompatibilities with
ANSI ANSI X3.135-1992
ISO ISO/IEC 9075:1992
(colons preceding <parameter name>s, WITH CHECK OPTION constraint clarifications), and aids for transitioning from
ANSI ANSI X3.135-1992
ISO ISO/IEC 9075:1992
to this
ANSI American
ISO International
Standard (<delimited identifier>s). Finally, it contains changes to correct defects found in
ANSI ANSI X3.135-1989
ISO ISO/IEC 9075:1989
(see Annex F, "Maintenance and interpretation of SQL").

Intermediate SQL includes major new facilities such as statements for changing schemas, isolation levels for SQL-transactions and named constraints. It also includes multiple-module support and cascade delete on referential actions, as well as numerous functional enhancements such as row and table expressions, union join, character string operations, table intersection and difference operations, simple domains, the CASE expression, casting between data types, a diagnostics management capability for data administration and more comprehensive error analysis, multiple character repertoires, interval and simplified datetime data types, and variable-length character strings. It also includes a requirement for a flagger facility to aid in writing portable applications.

Full SQL increases orthogonality and includes deferred constraint checking. Other technical enhancements include additional user options to define datetime data types, self-referencing updates and deletes, cascade update on referential actions, subqueries in check constraints, scrolled cursors, character translations, a bit string data type, temporary tables, additional referential constraint options, and simple assertions.

****Editor's Note****

This Subclause also must address the contents of the highest Level of SQL3, which should be given a proper name.

4.45 SQL Flagger

An SQL Flagger is an implementation-provided facility that is able to identify SQL language extensions, or other SQL processing alternatives, that may be provided by a conforming SQL-implementation (see Subclause 20.3, "Extensions and options"). An SQL Flagger is intended to assist SQL programmers in producing SQL language that is both portable and interoperable among different conforming SQL-implementations operating under different levels of this
ANSI American
ISO International
Standard.

An SQL Flagger is intended to effect a static check of SQL language. There is no requirement to detect extensions that cannot be determined until the General Rules are evaluated.

An SQL-implementation need only flag SQL language that is not otherwise in error as far as that implementation is concerned.

Note: If a system is processing SQL language that contains errors, then it may be very difficult within a single statement to determine what is an error and what is an extension. As one possibility, an implementation may choose to check SQL language in two steps; first through its normal syntax analyzer and secondly through the flagger. The first step produces error messages for nonstandard SQL language that the implementation cannot process or recognize. The second step processes SQL language that contains no errors as far as that implementation is concerned; it detects and flags at one time all nonstandard SQL language that could be processed by that implementation. Any such two-step process should be transparent to the end user.

In order to provide upward compatibility for its own customer base, or to provide performance advantages under special circumstances, a conforming SQL-implementation may provide user options to process conforming SQL language in a nonconforming manner. If this is the case, then it is required that the implementation also provide a flagger option, or some other implementation-defined means, to detect SQL conforming language that may be processed differently under the various user options. This flagger feature allows an application programmer to identify conforming SQL language that may perform differently in alternative processing environments provided by a conforming SQL-implementation. It also provides a valuable tool in identifying SQL elements that may have to be modified if SQL language is to be moved from a nonconforming to a conforming SQL processing environment.

An SQL Flagger provides one or more of the following “level of flagging” options:

— Entry SQL Flagging

— Intermediate SQL Flagging

— Full SQL Flagging

An SQL Flagger that provides one of these options shall be able to identify SQL language constructs that violate the indicated level of SQL language as defined in Subclause 4.44, "Leveling".

An SQL Flagger provides one or more of the following “extent of checking” options:

— Syntax Only

— Catalog Lookup

Under the Syntax Only option, the SQL Flagger analyzes only the SQL language that is presented; it checks for violations of any Syntax Rules that can be determined without access to the Information Schema.

Under the Catalog Lookup option, the SQL Flagger assumes the availability of Definition Schema information and checks for violations of all Syntax Rules and Access Rules, except Access Rules that deal with privileges. For example, some Syntax Rules place restrictions on data types; this flagger option would identify extensions that relax such restrictions. In order to avoid security breaches, this option may view the Definition Schema only through the eyes of a specific Information Schema.

5 Lexical elements

5.1 <SQL terminal character>

Function

Define the terminal symbols of the SQL language and the elements of strings.

Format

```
<SQL terminal character> ::=
      <SQL language character>

<SQL language character> ::=
        <simple Latin letter>
      | <digit>
      | <SQL special character>

<simple Latin letter> ::=
        <simple Latin upper case letter>
      | <simple Latin lower case letter>

<simple Latin upper case letter> ::=
            A | B | C | D | E | F | G | H | I | J | K | L | M | N | O
      | P | Q | R | S | T | U | V | W | X | Y | Z

<simple Latin lower case letter> ::=
            a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
      | p | q | r | s | t | u | v | w | x | y | z

<digit> ::=
      0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<SQL special character> ::=
        <space>
      | <double quote>
      | <percent>
      | <ampersand>
      | <quote>
      | <left paren>
      | <right paren>
      | <asterisk>
      | <plus sign>
      | <comma>
      | <minus sign>
      | <period>
      | <solidus>
      | <colon>
      | <semicolon>
      | <less than operator>
      | <equals operator>
      | <greater than operator>
      | <question mark>
      | <left bracket>
      | <right bracket>
```

```
       | <circumflex>
       | <underscore>
       | <vertical bar>

<space> ::= !! space character in character set in use

<double quote> ::= "

<percent> ::= %

<ampersand> ::= &

<quote> ::= '

<left paren> ::= (

<right paren> ::= )

<asterisk> ::= *

<plus sign> ::= +

<comma> ::= ,

<minus sign> ::= -

<period> ::= .

<solidus> ::= /

<colon> ::= :

<semicolon> ::= ;

<less than operator> ::= <

<equals operator> ::= =

<greater than operator> ::= >

<question mark> ::= ?

<left bracket> ::= [

<right bracket> ::= ]

<circumflex> ::= ^

<underscore> ::= _

<vertical bar> ::= |
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) There is a one-to-one correspondence between the symbols contained in <simple Latin upper case letter> and the symbols contained in <simple Latin lower case letter> such that, for all *i*, the symbol defined as the *i*-th alternative for <simple Latin upper case letter> corresponds to the symbol defined as the *i*-th alternative for <simple Latin lower case letter>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

5.2 <token> and <separator>

Function

Specify lexical units (tokens and separators) that participate in SQL language.

Format

```
<token> ::=
         <nondelimiter token>
       | <delimiter token>

<nondelimiter token> ::=
         <regular identifier>
       | <key word>
       | <unsigned numeric literal>
       | <national character string literal>
       | <bit string literal>
       | <hex string literal>
       | <user-defined operator symbol>

<regular identifier> ::= <identifier body>

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

<identifier start> ::= !! See the Syntax Rules

<identifier part> ::=
         <identifier start>
       | <digit>

<delimited identifier> ::=
       <double quote> <delimited identifier body> <double quote>

<delimited identifier body> ::= <delimited identifier part>...

<delimited identifier part> ::=
         <nondoublequote character>
       | <doublequote symbol>

<nondoublequote character> ::= !! See the Syntax Rules

<doublequote symbol> ::= <double quote><double quote>

<user-defined operator symbol> ::= !! See the Syntax Rules

<delimiter token> ::=
         <character string literal>
       | <date string>
       | <time string>
       | <timestamp string>
       | <interval string>
       | <delimited identifier>
       | <SQL special character>
       | <not equals operator>
       | <greater than or equals operator>
       | <less than or equals operator>
       | <concatenation operator>
```

```
       | <double period>
       | <double colon>
       | <assignment operator>
       | <left bracket>
       | <right bracket>

<not equals operator> ::= <>

<greater than or equals operator> ::= >=

<less than or equals operator> ::= <=

<concatenation operator> ::= ||

<double period> ::= ..

<double colon> ::= ::

<assignment operator> ::= :=

<keyword parameter tag> ::= =>

<separator> ::= { <comment> | <space> | <newline> }...

<comment> ::=
         <simple comment>
       | <bracketed comment>

<simple comment> ::=
       <simple comment introducer> [ <comment character>... ] <newline>

<simple comment introducer> ::= <minus sign><minus sign>[<minus sign>...]

<bracketed comment> ::=             !! (See the Syntax Rules)
       <bracketed comment introducer>
         <bracketed comment contents>
       <bracketed comment terminator>

<bracketed comment introducer> ::= <solidus><asterisk>

<bracketed comment terminator> ::= <asterisk><solidus>

<bracketed comment contents> ::=
       [ { <comment character> | <separator> }... ]

<comment character> ::=
         <nonquote character>
       | <quote>

<newline> ::= !! implementation-defined end-of-line indicator

<key word> ::=
         <reserved word>
       | <non-reserved word>

<non-reserved word> ::=
         ADA

       | C | CATALOG_NAME | CHAIN
```

```
        | CHARACTER_SET_CATALOG | CHARACTER_SET_NAME
        | CHARACTER_SET_SCHEMA | CLASS_ORIGIN | COBOL | COLLATION_CATALOG | COLLATION_NAME

        | COLLATION_SCHEMA | COLUMN_NAME | COMMAND_FUNCTION
        | COMMAND_FUNCTION_CODE
        | COMMITTED | CONDITION_NUMBER | CONNECTION_NAME | CONSTRAINT_CATALOG
        | CONSTRAINT_NAME | CONSTRAINT_SCHEMA | CURSOR_NAME

        | DATA | DATETIME_INTERVAL_CODE | DATETIME_INTERVAL_PRECISION | DYNAMIC_FUNCTION
        | DYNAMIC_FUNCTION_CODE

        | FORTRAN

        | HOLD

        | INFIX
        | KEY_MEMBER | KEY_TYPE

        | LENGTH

        | MESSAGE_LENGTH | MESSAGE_OCTET_LENGTH | MESSAGE_TEXT | MORE | MUMPS

        | NAME | NULLABLE | NUMBER

        | OPERATORS

        | PASCAL | PLI | POSTFIX | PREFIX

        | REPEATABLE | RETURNED_LENGTH | RETURNED_OCTET_LENGTH | RETURNED_SQLSTATE
        | ROUTINE_CATALOG | ROUTINE_NAME | ROUTINE_SCHEMA | ROW_COUNT

        | SCALE | SCHEMA_NAME | SERIALIZABLE | SERVER_NAME | SPECIFIC_NAME
        | SUBCLASS_ORIGIN

        | TABLE_NAME
        | TRIGGER_CATALOG | TRIGGER_SCHEMA | TRIGGER_NAME
        | TRANSACTIONS_COMMITTED | TRANSACTIONS_ROLLED_BACK | TRANSACTION_ACTIVE

        | UNCOMMITTED | UNNAMED

<reserved word> ::=
          ABSOLUTE | ACTION | ACTOR | ADD
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
        | ADMIN
```

```
        | AFTER | ALIAS
        | ALL | ALLOCATE | ALTER | AND | ANY | ARE
        | AS | ASC | ASSERTION | ASYNC | AT
        | AUTHORIZATION | AVG

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

        | CASCADE | CASCADED | CASE | CAST | CATALOG | CHAR | CHARACTER | CHAR_LENGTH
        | CHARACTER_LENGTH | CHECK | CLASS | CLOB | CLOSE | COALESCE | COLLATE
        | COLLATION | COLUMN | COMMIT | COMPLETION | CONNECT | CONNECTION | CONSTRAINT
```

| CONSTRAINTS | CONSTRUCTOR | CONTINUE | CONVERT | CORRESPONDING | COUNT
| CREATE | CROSS | CURRENT | CURRENT_DATE
| CURRENT_PATH | CURRENT_TIME
| CURRENT_TIMESTAMP | CURRENT_USER | CURSOR | CYCLE

| DATA | DATE | DAY | DEALLOCATE | DEC | DECIMAL | DECLARE | DEFAULT
| DEFERRABLE | DEFERRED | DELETE | DEPTH | DESC | DESCRIBE
| DESCRIPTOR | DESTROY | DESTRUCTOR | DICTIONARY
| DIAGNOSTICS | DISCONNECT | DISTINCT | DOMAIN | DOUBLE | DROP

| EACH

ANSI Only---caused by ISO changes not yet considered by ANSI

| ELEMENT

| ELSE
| END | END-EXEC | EQUALS | ESCAPE | EXCEPT
| EXEC | EXECUTE | EXISTS | EXTERNAL | EXTRACT | FALSE | FETCH | FIRST | FLOAT | FOR | FOREIGN
| FOUND | FROM | FREE | FULL
| FUNCTION

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

| HAVING | HASH | HOST | HOUR

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

| JOIN

| KEY

| LANGUAGE | LARGE | LAST | LEADING | LEFT | LESS | LEVEL | LIKE | LIMIT

ANSI Only--SQL3

| LIST

| LOCAL | LOCATOR | LOWER

| MATCH | MAX | MIN | MINUTE | MODIFY | MODULE | MONTH

ANSI Only---caused by ISO changes not yet considered by ANSI

| MOVE | MULTISET

| NAMES | NATIONAL | NATURAL | NCHAR | NCLOB | NEW_TABLE
| NEXT | NO
| NONE | NOT| NULL | NULLIF | NUMERIC

| OBJECT | OCTET_LENGTH | OF | OFF | OID | OLD | OLD_TABLE

```
| ON | ONLY | OPEN | OPERATION | OPERATORS
| OPTION | OR | ORDER | OUT | OUTER | OUTPUT | OVERLAPS | OVERLAY

| PAD | PARAMETERS | PARTIAL | PATH
| PENDANT | POSITION | POSTFIX | PRECISION | PREFIX
| PREORDER | PREPARE | PRESERVE | PRIMARY | PRIOR | PRIVATE
| PRIVILEGES | PROCEDURE | PROTECTED | PUBLIC

| READ | REAL | RECURSIVE
| REFERENCES | REFERENCING | RELATIVE | REPRESENTATION | RESTRICT | REVOKE | RIGHT

| ROLE | ROLLBACK | ROUTINE | ROW | ROWS

| SAVEPOINT | SCHEMA | SCROLL | SEARCH | SECOND | SECTION | SELECT
| SENSITIVE | SEQUENCE | SESSION | SESSION_USER | SET
| SIMILAR | SIZE | SMALLINT | SOME | SPACE | SPECIFIC | SQL | SQLCODE
| SQLERROR | SQLEXCEPTION | SQLSTATE | SQLWARNING | START | STATE
| STRUCTURE | SUBSTRING | SUM | SYSTEM_USER

| TABLE | TEMPLATE | TEMPORARY | TEST | THAN
| THEN
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
| THERE
```

```
| TIME | TIMESTAMP | TIMEZONE_HOUR
| TIMEZONE_MINUTE | TO | TRAILING | TRANSACTION | TRANSLATE | TRANSLATION
| TREAT | TRIGGER | TRIM | TRUE | TYPE

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

| VALUE | VALUES | VARCHAR | VARIABLE | VARIANT | VARYING | VIEW | VISIBLE

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

| YEAR

| ZONE
```

Note: The list of <reserved word>s is considerably longer than the analogous list of <key word>s in
ANSI X3.135-1992.
ISO ISO 9075:1992.
To assist users of this
ANSI American
ISO International
Standard avoid such words in a possible future revision, the following list of potential <reserved word>s is provided. Readers must understand that there is no guarantee that all of these words will, in fact, become <reserved word>s in any future revision; furthermore, it is almost certain that additional words will be added to this list as any possible future revision emerges.

The words are: To Be Supplied

Syntax Rules

1) An <identifier start> is one of:

   a) A <simple Latin letter>; or

   b) A character that is identified as a letter in the character repertoire identified by the <module character set specification> or by the <character set specification>; or

   c) A character that is identified as a syllable in the character repertoire identified by the <module character set specification> or by the <character set specification>; or

   d) A character that is identified as an ideograph in the character repertoire identified by the <module character set specification> or by the <character set specification>.

2) With the exception of the <space> character explicitly contained in <timestamp string> and <interval string> and the permitted <separator>s in <bit string literal>s and <hex string literal>s, a <token>, other than a <character string literal>, a <national character string literal>, or a <delimited identifier>, shall not include a <space> character or other <separator>.

3) A <nondoublequote character> is one of:

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

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

   c) Any character other than a <double quote> in the character repertoire identified by the <character set specification>.

4) The two <doublequote>s contained in a <doublequote symbol> shall not be separated by any <separator>.

5) Any <token> may be followed by a <separator>. A <nondelimiter token> shall be followed by a <delimiter token> or a <separator>.

   Note: If the Format does not allow a <nondelimiter token> to be followed by a <delimiter token>, then that <nondelimiter token> shall be followed by a <separator>.

6) There shall be no <space> nor <newline> separating the <minus sign>s of a <comment introducer>.

7) There shall be no <separator> between the <period>s of a <double period>.

8) There shall be no <separator> between the <solidus> and <asterisk> of a <bracketed comment introducer> or between the <asterisk> and <solidus> of a <bracketed comment terminator>.

9) Within a <bracketed comment contents>, any <solidus> immediately followed by an <asterisk> without any intervening <separator> shall be considered to be the <bracketed comment introducer> of a <separator> that is a <bracketed comment>.

   Note: Conforming programs should not place <simple comment> within a <bracketed comment> because if such a <simple comment> contains the sequence of characters "*/" without a preceding "/*" in the same <simple comment>, it will prematurely terminate the containing <bracketed comment>.

10) SQL text containing one or more instances of <comment> is equivalent to the same SQL text with the <comment> replaced with <newline>.

11) The sum of the number of <identifier start>s and the number of <identifier part>s in a <regular identifier> shall not be greater than 128.

12) The <delimited identifier body> of a <delimited identifier> shall not comprise more than 128 <delimited identifier part>s.

13) The <identifier body> of a <regular identifier> is equivalent to an <identifier body> in which every letter that is a lower-case letter is replaced by the equivalent upper-case letter or letters. This treatment includes determination of equivalence, representation in the Information and Definition Schemas, representation in the diagnostics area, and similar uses.

14) The <identifier body> of a <regular identifier> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT, shall not be equal, according to the comparison rules in Subclause 8.2, "<comparison predicate>", to any <reserved word> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT.

Note: It is the intention that no <key word> specified in this
ANSI American
ISO International
Standard or revisions thereto shall end with an <underscore>.

15) Two <regular identifier>s are equivalent if their <identifier body>s, considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT, compare equally according to the comparison rules in Subclause 8.2, "<comparison predicate>".

16) A <regular identifier> and a <delimited identifier> are equivalent if the <identifier body> of the <regular identifier> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters) and the <delimited identifier body> of the <delimited identifier> (with all occurrences of <quote> replaced by <quote symbol> and all occurrences of <double-quote symbol> replaced by <double quote>), considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT and an implementation-defined collation that is sensitive to case, compare equally according to the comparison rules in Subclause 8.2, "<comparison predicate>".

17) Two <delimited identifier>s are equivalent if their <delimited identifier body>s (with all occurrences of <doublequote symbol> replaced by <doublequote>), considered as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT and an implementation-defined collation that is sensitive to case, compare equally according to the comparison rules in Subclause 8.2, "<comparison predicate>".

18) For the purposes of identifying <key word>s, any <simple Latin lower case letter> contained in a candidate <key word> shall be effectively treated as the corresponding <simple Latin upper case letter>.

19) Case:

    a) If <token> is contained in a <schema definition> *SD*, then let *SN* be the <schema name> of *SD*.

    b) If <token> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement>, or contained in a <direct SQL statement> that is invoked directly, then let *SN* be a <schema name>

containing the default unqualified schema name and the default catalog name of the SQL-session.

c) Otherwise, let *SN* be the <schema name> that is specified or implicit for the <module>.

20) A <user-defined operator symbol> is an <operator symbol> that is contained in an <operator group> in *OD*.

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) No <identifier body> shall end in an <underscore>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) No <regular identifier> or <delimited identifier body> shall contain more than 18 <character representation>s.

b) An <identifier body> shall contain no <simple Latin lower case letter>.

5.3 <literal>

Function

Specify a non-null value.

Format

```
<literal> ::=
         <signed numeric literal>
       | <general literal>

<unsigned literal> ::=
         <unsigned numeric literal>
       | <general literal>

<general literal> ::=
         <character string literal>
       | <national character string literal>
       | <bit string literal>
       | <hex string literal>
       | <binary string literal>
       | <datetime literal>
       | <interval literal>
       | <enumeration literal>
       | <boolean literal>
       | <oid literal>

<character string literal> ::=
       [ <introducer><character set specification> ]
       <quote> [ <character representation>... ] <quote>
         [ { <separator> <quote> [ <character representation>... ] <quote> }... ]

<introducer> ::= <underscore>

<character representation> ::=
         <nonquote character>
       | <quote symbol>

<nonquote character> ::= !! See the Syntax Rules.

<quote symbol> ::= <quote><quote>

<national character string literal> ::=
       N <quote> [ <character representation>... ] <quote>
         [ { <separator> <quote> [ <character representation>... ] <quote> }... ]

<bit string literal> ::=
       B <quote> [ <bit>... ] <quote>
         [ { <separator> <quote> [ <bit>... ] <quote> }... ]

<hex string literal> ::=
       X <quote> [ <hexit>... ] <quote>
         [ { <separator> <quote> [ <hexit>... ] <quote> }... ]

<bit> ::= 0 | 1

<hexit> ::= <digit> | A | B | C | D | E | F | a | b | c | d | e | f
```

```
<signed numeric literal> ::=
       [ <sign> ] <unsigned numeric literal>

<unsigned numeric literal> ::=
         <exact numeric literal>
       | <approximate numeric literal>

<exact numeric literal> ::=
         <unsigned integer> [ <period> [ <unsigned integer> ] ]
       | <period> <unsigned integer>

<sign> ::= <plus sign> | <minus sign>

<approximate numeric literal> ::= <mantissa> E <exponent>

<mantissa> ::= <exact numeric literal>

<exponent> ::= <signed integer>

<signed integer> ::= [ <sign> ] <unsigned integer>

<unsigned integer> ::= <digit>...

<enumeration literal> ::=
       <domain name> <double colon> <enumeration name>

<datetime literal> ::=
         <date literal>
       | <time literal>
       | <timestamp literal>

<date literal> ::=
       DATE <date string>

<time literal> ::=
       TIME <time string>

<timestamp literal> ::=
       TIMESTAMP <timestamp string>

<date string> ::=
       <quote> <unquoted date string> <quote>

<time string> ::=
       <quote> <unquoted time string> <quote>

<timestamp string> ::=
       <quote> <unquoted timestamp string> <quote>

<time zone interval> ::=
       <sign> <hours value> <colon> <minutes value>

<date value> ::=
       <years value> <minus sign> <months value> <minus sign> <days value>

<time value> ::=
       <hours value> <colon> <minutes value> <colon> <seconds value>

<interval literal> ::=
```

```
      INTERVAL [ <sign> ] <interval string> <interval qualifier>

<interval string> ::=
      <quote> <unquoted interval string> <quote>



<unquoted date string> ::= <date value> <unquoted time string> ::=
      <time value> [ <time zone interval> ]

<unquoted timestamp string> ::=
      <unquoted date string> <space> <unquoted time string>

<unquoted interval string> ::=
      [ <sign> ] { <year-month literal> | <day-time literal> }

<unquoted temporal string> ::=
        <unquoted date string>
      | <unquoted time string>
      | <unquoted timestamp string>
      | <unquoted interval string>

<year-month literal> ::=
        <years value>
      | [ <years value> <minus sign> ] <months value>

<day-time literal> ::=
        <day-time interval>
      | <time interval>

<day-time interval> ::=
      <days value>
        [ <space> <hours value> [ <colon> <minutes value> [ <colon> <seconds value> ] ] ]


<time interval> ::=
        <hours value> [ <colon> <minutes value> [ <colon> <seconds value> ] ]
      | <minutes value> [ <colon> <seconds value> ]
      | <seconds value>

<years value> ::= <datetime value>

<months value> ::= <datetime value>

<days value> ::= <datetime value>

<hours value> ::= <datetime value>

<minutes value> ::= <datetime value>

<seconds value> ::=
        <seconds integer value> [ <period> [ <seconds fraction> ] ]

<seconds integer value> ::= <unsigned integer>

<seconds fraction> ::= <unsigned integer>

<datetime value> ::= <unsigned integer>

<boolean literal> ::=
```

```
        TRUE
      | FALSE

<oid literal> ::= OID <oid string>

<oid string> ::= <quote> <oid value> <quote>

<oid value> ::= <character representation>...

<binary string litera > ::=
      BIN <quote> [ { <hexit> <hexit> }... ] <quote>
        [ { <separator> <quote> [ { <hexit> <hexit> }... ] <quote> }... ]
```

Syntax Rules

1) In a <character string literal> or <national character string literal>, the sequence:

   <quote> <character representation>... <quote>
   <separator> <quote> <character representation>... <quote>

   is equivalent to the sequence

   <quote> <character representation>... <character representation>... <quote>

   Note: The <character representation>s in the equivalent sequence are in the same sequence and relative sequence as in the original <character string literal>.

2) In a <bit string literal>, the sequence

   <quote> <bit>... <quote> <separator> <quote> <bit>... <quote>

   is equivalent to the sequence

   <quote> <bit>... <bit>... <quote>

   Note: The <bit>s in the equivalent sequence are in the same sequence and relative sequence as in the original <bit string literal>.

3) In a <hex string literal>, the sequence

   <quote> <hexit>... <quote> <separator> <quote> <hexit>... <quote>

   is equivalent to the sequence

   <quote> <hexit>... <hexit>... <quote>

   Note: The <hexit>s in the equivalent sequence are in the same sequence and relative sequence as in the original <hex string literal>.

4) In a <character string literal>, <national character string literal>, <bit string literal>, <binary string literal>, or <hex string literal>, a <separator> shall contain a <newline>.

5) A <nonquote character> is one of:

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

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

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

6) If a <character set specification> is not specified in a <character string literal>, then the set of characters contained in the <character string literal> shall be wholly contained in either <SQL language character> or the character repertoire indicated by:

   Case:

   a) If the <character string literal> is contained in a <module>, then the <module character set specification>,

   b) If the <character string literal> is contained in a <schema definition> that is not contained in a <module>, then the <schema character set specification>,

7) If a <character set specification> is specified in a <character string literal>, then

   a) There shall be no <separator> between the <introducer> and the <character set specification>.

   b) The set of characters contained in the <character string literal> shall be wholly contained in the character repertoire indicated by the <character set specification>.

8) A <national character string literal> is equivalent to a <character string literal> with the "N" replaced by "<introducer><character set specification>", where "<character set specification>" is an implementation-defined <character set name>.

9) The data type of a <character string literal> is fixed-length character string. The length of a <character string literal> is the number of <character representation>s that it contains. Each <quote symbol> contained in <character string literal> represents a single <quote> in both the value and the length of the <character string literal>. The two <quote>s contained in a <quote symbol> shall not be separated by any <separator>.

   Note: <character string literal>s are allowed to be zero-length strings (*i.e.*, to contain no characters) even though it is not permitted to declare a <data type> that is CHARACTER with <length> zero.

10) The data type of a <bit string literal> is fixed-length bit string. The length of a <bit string literal> is the number of bits that it contains.

11) The data type of a <hex string literal> is fixed-length bit string. Each <hexit> appearing in the literal is equivalent to a quartet of bits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F are interpreted as 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111, respectively. The <hexit>s a, b, c, d, e, and f have respectively the same values as the <hexit>s A, B, C, D, E, and F.

12) An <exact numeric literal> without a <period> has an implied <period> following the last <digit>.

13) The data type of an <exact numeric literal> is exact numeric. The precision of an <exact numeric literal> is the number of <digit>s that it contains. The scale of an <exact numeric literal> is the number of <digit>s to the right of the <period>.

14) The data type of an <approximate numeric literal> is approximate numeric. The precision of an <approximate numeric literal> is the precision of its <mantissa>.

15) The <domain name> of an <enumeration literal> shall identify a <domain definition> whose <data type> is an <enumerated type> that includes the <enumeration name> of the <enumeration literal>. That <domain> and <data type> are the <domain> and <data type> of the <enumeration literal>.

16) The data type of a <date literal> is DATE.

17) The data type of a <time literal> that does not specify <time zone interval> is TIME(*P*), where *P* is the number of digits in <seconds fraction>, if specified, and 0 otherwise. The data type of a <time literal> that specifies <time zone interval> is TIME(*P*) WITH TIME ZONE, where *P* is the number of digits in <seconds fraction>, if specified, and 0 otherwise.

18) The data type of a <timestamp literal> that does not specify <time zone interval> is TIMESTAMP(*P*), where *P* is the number of digits in <seconds fraction>, if specified, and 0 otherwise. The data type of a <timestamp literal> that specifies <time zone interval> is TIMESTAMP(*P*) WITH TIME ZONE, where *P* is the number of digits in <seconds fraction>, if specified, and 0 otherwise.

19) If <time zone interval> is not specified, then the effective <time zone interval> of the datetime data type is the current default time zone displacement for the SQL-session.

20) Let *datetime component* be either <years value>, <months value>, <days value>, <hours value>, <minutes value>, or <seconds value>.

21) Let *N* be the number of <datetime field>s in the precision of the <interval literal>, as specified by <interval qualifier>.

    The <interval literal> being defined shall contain *N* datetime components.

    The data type of <interval literal> specified with an <interval qualifier> is INTERVAL with the <interval qualifier>.

22) Within a <datetime literal>, the <years value> shall contain four digits. The <seconds integer value> and other datetime components, with the exception of <seconds fraction>, shall each contain two digits.

23) Within the definition of a <datetime literal>, the <datetime value>s are constrained by the natural rules for dates and times according to the Gregorian calendar.

24) Within the definition of an <interval literal>, the <datetime value>s are constrained by the natural rules for intervals according to the Gregorian calendar.

25) Within the definition of a <year-month literal>, the <interval qualifier> shall not specify DAY, HOUR, MINUTE, or SECOND. Within the definition of a <day-time literal>, the <interval qualifier> shall not specify YEAR or MONTH.

26) Within the definition of a <datetime literal>, the value of the <time zone interval> shall be in the range −12:59 to +13:00.

27) The null class of a <literal> is the general null class.

28) The <oid value> of an <oid literal> represents an object identifier. Each <quote symbol> contained in an <oid value> represents a single <quote> in the value. The two <quote>s contained in a <quote symbol> shall not be separated by any <separator>. The method of representing an object identifier as an <oid value> is implementation-dependent.

29) The data type of an <oid literal> is the data type of the abstract data type that its <oid value> represents.

30) Within the definition of an <oid literal>, the allowed <oid value>s are implementation-dependent.

31) In a <binary string literal>, the sequence

    <quote> { <hexit> <hexit> }... <quote> <separator> <quote> { <hexit> <hexit> }... <quote>

    is equivalent to the sequence

    <quote> { <hexit> <hexit> }... { <hexit> <hexit> }... <quote>

    Note: The <hexits> in the equivalent sequence are in the same sequence and relative sequence as in the original <binary string literal>.

32) The data type of a <binary string literal> is a binary string. Each <hexit> appearing in the literal is equivalent to a quartet of bits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F are interpreted as 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111, respectively. The <hexit>s a, b, c, d, e, and f have respectively the same values as the <hexit>s A, B, C, D, E, and F.

33) A <binary string literal> shall have an even number of <hexits>.

Access Rules

None.

General Rules

1) The value of a <character string literal> is the sequence of <character representation>s that it contains.

2) The value of a <bit string literal> or a <hex string literal> is the sequence of bits that it contains.

3) The numeric value of an <exact numeric literal> is determined by the normal mathematical interpretation of positional decimal notation.

4) The numeric value of an <approximate numeric literal> is approximately the product of the exact numeric value represented by the <mantissa> with the number obtained by raising the number 10 to the power of the exact numeric value represented by the <exponent>.

5) The <sign> in a <signed numeric literal> or an <interval literal> is a monadic arithmetic operator. The monadic arithmetic operators + and − specify monadic plus and monadic minus, respectively. If neither monadic plus nor monadic minus are specified in a <signed numeric literal> or an <interval literal>, or if monadic plus is specified, then the literal is positive. If monadic minus is specified in a <signed numeric literal> or <interval literal>, then the literal is negative. If a <sign> is specified in both possible locations in an <interval literal>, then the sign of the literal is determined by the normal mathematical interpretation of multiple sign operators.

6) The value of an <enumeration literal> is its enumeration value.

7) Let V be the integer value of the <unsigned integer> contained in <seconds fraction> and let N be the number of digits in the <seconds fraction> respectively. The resultant value of the <seconds fraction> is effectively determined as follows:

   Case:

   a) If <seconds fraction> is specified within the definition of a <datetime literal>, then the effective value of the <seconds fraction> is $V * 10^{-N}$ seconds.

b) If <seconds fraction> is specified within the definition of an <interval literal>, then let M be the <interval fractional seconds precision> specified in the <interval qualifier>.

Case:

i) If $N < M$, then let $V1$ be $V * 10^{M-N}$; the effective value of the <seconds fraction> is $V1 * 10^{-M}$ seconds.

ii) If $N > M$, then let $V2$ be the integer part of the quotient of $V/10^{N-M}$; the effective value of the <seconds fraction> is $V2 * 10^{-M}$ seconds.

iii) Otherwise, the effective value of the <seconds fraction> is $V * 10^{-M}$ seconds.

8) The i-th datetime component in a <datetime literal> or <interval literal> assigns the value of the datetime component to the i-th <datetime field> in the <datetime literal> or <interval literal>.

9) If <time zone interval> is specified, then the time and timestamp values in <time literal> and <timestamp literal> represent a datetime in the specified time zone. Otherwise, the time and timestamp values represent a datetime in the current default time zone of the SQL-session. The value of the <time literal> or the <timestamp literal> is effectively the <time value> or the <date value> and <time value> together minus the <time zone interval> value, followed by the <time zone interval>.

Note: <time literal>s and <timestamp literal>s are specified in a time zone chosen by the SQL-agent (the default is the current default time zone of the SQL-session). However, they are effectively converted to UTC while maintaining the <time zone interval> information that permits knowing the original time zone value for the time or timestamp value.

10) The value of an <oid literal> is that instance of the abstract data type identified by the <oid value> of that <oid literal>.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <general literal> shall not be an <enumeration literal>.

b) A <general literal> shall not be a <boolean literal>.

c) A <general literal> shall not be an <oid literal>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) An <unsigned integer> that is a <seconds fraction> shall not contain more than 6 <digit>s.

b) A <general literal> shall not be a <bit string literal> or a <hex string literal>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <general literal> shall not be a <national character string literal>.

b) A <general literal> shall not be a <datetime literal> or <interval literal>.

c) A <character string literal> shall contain at least one <character representation>.

d) Conforming Entry SQL language shall contain exactly one repetition of <character representation> (that is, it shall contain exactly one sequence of "<quote> <character representation>... <quote>").

e) A <character string literal> shall not specify a <character set specification>.

5.4 Names and identifiers

Function

Specify names.

Format

```
<identifier> ::=
       [ <introducer><character set specification> ] <actual identifier>

<actual identifier> ::=
         <regular identifier>
       | <delimited identifier>

<SQL language identifier> ::=
       <SQL language identifier start>
         [ { <underscore> | <SQL language identifier part> }... ]

<SQL language identifier start> ::= <simple Latin letter>

<SQL language identifier part> ::=
         <simple Latin letter>
       | <digit>

<authorization identifier> ::= <identifier>

<table name> ::=
         <local or schema qualified name>

<domain name> ::= <schema qualified name>

<enumeration name> ::= <identifier>

<schema name> ::=
       [ <catalog name> <period> ] <unqualified schema name>

<unqualified schema name> ::= <identifier>

<catalog name> ::= <identifier>

<schema qualified name> ::=
       [ <schema name> <period> ] <qualified identifier>

<local or schema qualified name> ::=
       [ <local or schema qualifier> <period> ] <qualified identifier>

<local or schema qualifier> ::=
         <schema name>
       | MODULE

<qualified identifier> ::= <identifier>

<column name> ::=
         <identifier>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
      | OID
```

ANSI Only--SQL3

```
      | IDENTITY
```

```
<correlation name> ::= <identifier>
```

ISO Only--SQL3

```
<query name> ::= <identifier>
```

```
<module name> ::=
        <SQL-server module name>
      | <SQL-client module name>

<SQL-server module name> ::= <qualified identifier>

<SQL-client module name> ::= <identifier>

<cursor name> ::= <local qualified name>

<local qualified name> ::=
      [ <local qualifier> <period> ] <qualified identifier>

<local qualifier> ::= MODULE
```

- 2 productions moved to Part 4

```
<parameter name> ::= [ <colon> ] <identifier>

<type template name> ::= <schema qualified name>

<template parameter name> ::= <colon> <identifier>

<constraint name> ::= <schema qualified name>

<external routine name> ::=
        <identifier>
      | <character string literal>

<trigger name> ::= <schema qualified name>

<collation name> ::= <schema qualified name>

<character set name> ::= [ <schema name> <period> ] <SQL language identifier>

<translation name> ::= <schema qualified name>

<form-of-use conversion name> ::= <schema qualified name>
```

```
<abstract data type name> ::= <local or schema qualified name> <attribute name> ::=
        <identifier>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
| OID
```

```
<component name> ::= <identifier>

<operator name> ::= <identifier>

<field name> ::= <identifier>

<savepoint name> ::= <identifier>

<distinct type name> ::= <schema qualified name>

<role name> ::= <authorization identifier>

<connection name> ::= <simple value specification>

<null class name> ::= <schema qualified name>

<SQL-server name> ::= <simple value specification>

<user name> ::= <simple value specification>
```

Syntax Rules

1) If a <character set specification> is not specified in an <identifier>, then the set of characters contained in the <identifier> shall be wholly contained in either <SQL language character> or the character repertoire indicated by:

   Case:

   a) If the <identifier> is contained in a <module>, then the <module character set specification>,

   b) If the <identifier> is contained in a <schema definition> that is not contained in a <module>, then the <schema character set specification>,

2) If a <character set specification> is specified in an <identifier>, then:

   a) There shall be no <separator> between the <introducer> and the <character set specification>.

   b) The set of characters contained in the <identifier body> or <delimited identifier body> shall be wholly contained in the character repertoire indicated by the <character set specification>.

3) The sum of the number of <SQL language identifier start>s and the number of <SQL language identifier part>s in an <SQL language identifier> shall not be greater than 128.

4) An <SQL language identifier> is equivalent to an <SQL language identifier> in which every letter that is a lower-case letter is replaced by the equivalent upper-case letter or letters. This treatment includes determination of equivalence, representation in the Information and Definition Schemas, representation in the diagnostics area, and similar uses.

5) An <SQL language identifier> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT, shall not be equal, according to the comparison rules in Subclause 8.2, "<comparison predicate>", to any <reserved word> (with every letter that is a lower-case letter replaced by the corresponding upper-case letter or letters), treated as the repetition of a <character string literal> that specifies a <character set specification> of SQL_TEXT.

   Note: It is the intention that no <key word> specified in this
   ANSI American
   ISO International
   standard or revisions thereto shall end with an <underscore>.

6) If a <table name> *TN* with a <qualified identifier> *QI* does not contain a <local or schema qualifier>, then

   Case:

   a) If *TN* is contained in a <module> *M* whose <module contents> contain a <temporary table declaration> or <temporary view declaration> *TT* whose <table name> *TN* has a <qualified identifier> equal to *QI*, then the <module name> of *M* is the implicit <local or schema qualifier> of *TN*.

   b) Otherwise, *TN* shall be contained in a <schema definition> *S* that contains a <table definition> or <view definition> whose <table name> has a <qualified identifier> equal to *QI*. The <schema name> of *S* is the implicit <local or schema qualifier> of *TN*.

7) If a <table name> *TN* with a <qualified identifier> *QI* contains a <local or schema qualifier> *LSQ*, then

   Case:

   a) If *TN* is contained in a <module> *M* whose <module name> is *LSQ*, or if *LSQ* is “MODULE”, then the <module contents> of *M* shall contain a <temporary table declaration> or <temporary view declaration> *TT* whose <table name> has a <qualified identifier> equal to *QI*.

   b) Otherwise, *LSQ* shall be a <schema name> that designates a schema that contains a <table definition> or <view definition> whose <table name> has a <qualified identifier> equal to *QI*.

8) If a <cursor name> *CN* with a <qualified identifier> *QI* does not contain a <local qualifier>, then *CN* shall be contained in a <module> whose <module contents> contain a <declare cursor> whose <cursor name> *CN* has a <qualified identifier> equal to *QI* and “MODULE” is the implicit <local qualifier> of *CN*.

9) If a <cursor name> *CN* with a <qualified identifier> *QI* contains a <local qualifier> *LQ*, then *LQ* shall be “MODULE” and *CN* shall be contained in a <module> whose <module contents> contain a <declare cursor> whose <cursor name> is *CN*.

10) If an <abstract data type name> *ADTN* with a <qualified identifier> *QI* does not contain a <local or schema qualifier>, then

Case:

a) If *ADTN* is immediately contained in a <user-defined type>, then:

   i) An abstract data type T_i is a *possibly candidate type* if the <qualified identifier> of T_i is equal to *ADTN*.

   ii) The Syntax Rules of Subclause 9.5, "Subject type determination", are applied to *ADTN*, yielding an *identified candidate type* and the implicit <schema name>.

b) Otherwise,

   Case:

   i) If *ADTN* is contained in a <module> *M* whose <module contents> contain a <temporary abstract data type declaration> whose <abstract data type name> has a <qualified identifier> equal to *QI*, then the <module name> of *M* is the implicit <local or schema qualifier> of *ADTN*.

   ii) Otherwise, *ADTN* shall be contained in a schema *S* that contains an <abstract data type definition> whose <abstract data type name> has a <qualified identifier> equal to *QI*. The <schema name> of *S* is the implicit <local or schema qualifier> of *ADTN*.

11) If an <abstract data type name> *ADTN* with a <qualified identifier> *QI* contains a <local or schema qualifier> *LSQ*, then

Case:

a) If *ADTN* is contained in a <module> *M* whose <module name> is *LSQ*, or if *LSQ* is “MODULE”, then the <module contents> of *M* shall contain a <temporary abstract data type declaration> whose <abstract data type name> has a <qualified identifier> equal to *QI*.

b) Otherwise, *LSQ* shall be a <schema name> that designates a schema that contains an <abstract data type definition> whose <abstract data type name> has a <qualified identifier> equal to *QI*.

12) If a <distinct type name> *DTN* with a <qualified identifier> *QI* does not contain a <local or schema qualifier>, then

Case:

a) If *DTN* is immediately contained in a <user-defined type>, then:

   i) A distinct type T_i is a *possibly candidate type* if the <qualified identifier> of T_i is equal to *DTN*.

   ii) The Syntax Rules of Subclause 9.5, "Subject type determination", are applied to *DTN*, yielding an *identified candidate type* and the implicit <schema name>.

b) Otherwise, *DTN* shall be contained in a schema *S* that contains a <distinct type definition> whose <distinct type name> has a <qualified identifier> equal to *QI*. The <schema name> of *S* is the implicit <local or schema qualifier> of *DTN*.

13) If a <distinct type name> *DTN* with a <qualified identifier> *QI* contains a <local or schema qualifier> *LSQ*, then *LSQ* shall be a <schema name> that designates a schema that contains a <distinct type definition> whose <distinct type name> has a <qualified identifier> equal to *QI*.

14) Case:

a) If a <type template name> *TTN* with a <qualified identifier> *QI* is contained in a <user-defined type>, then:

i) A type template T_i is a *possibly candidate type* if the <qualified identifier> of T_i is equal to *TTN* and the number of <template parameter name>s contained in the <type template definition> of T_i is equal to the number of <template parameter>s immediately contained in <template parameter list> *TPL* of the <generated type reference> that immediately contains *TTN*.

ii) The Syntax Rules of Subclause 9.5, "Subject type determination", are applied to *TTN* and *TPL*, yielding an *identified candidate type* and the implicit <schema name>.

iii) Let *identified candidate <type template definition>* be the identified candidate type.

b) Otherwise,

Case:

i) If *TTN* does not contain a <local or schema qualifier>, then *TTN* shall be contained in a schema *S* that contains a <type template definition> whose <type template name> has a <qualified identifier> equal to *QI*. The <schema name> of *S* is the implicitl <local or schema qualifier> of *TTN*.

ii) If *TTN* contains a <local or schema qualifier> *LSQ*, then *LSQ* shall be a <schema name> that designates a schema that contains a <type template definition> whose <type template name> has a <qualified identifier> equal to *QI*.

15) No <unqualified schema name> shall specify DEFINITION_SCHEMA.

16) If a <schema qualified name> does not contain a <schema name>, then

Case:

a) If the <schema qualified name> is contained in a <schema definition>, then the <schema name> that is specified or implicit in the <schema definition> is implicit.

b) Otherwise, the <schema name> that is specified or implicit for the <module> is implicit.

17) If a <schema name> does not contain a <catalog name>, then

Case:

a) If the <unqualified schema name> is contained in a <module authorization clause>, then an implementation-defined <catalog name> is implicit.

b) If the <unqualified schema name> is contained in a <schema definition> other than in a <schema name clause>, then the <catalog name> that is specified or implicit in the <schema name clause> is implicit.

c) If the <unqualified schema name> is contained in a <schema name clause>, then

Case:

i) If the <schema name clause> is contained in a <module>, then the explicit or implicit <catalog name> contained in the <module authorization clause> is implicit.

ii) Otherwise, an implementation-defined <catalog name> is implicit.

d) Otherwise, the explicit or implicit <catalog name> contained in the <module authorization clause> is implicit.

18) Two <schema qualified name>s are equal if and only if they have the same <qualified identifier> and the same <schema name>, regardless of whether the <schema name>s are implicit or explicit.

19) Two <schema name>s are equal if and only if they have the same <unqualified schema name> and the same <catalog name>, regardless of whether the <catalog name>s are implicit or explicit.

20) An <identifier> that is a <correlation name> is associated with a table within a particular scope. The scope of a <correlation name> is either a <select statement: single row>, <subquery>, or <query specification> (see Subclause 6.7, "<table reference>"), or is a <query term> that contains a <recursive union> (see Subclause 7.15, "<recursive union>"), or is a <trigger definition> (see Subclause 11.45, "<trigger definition>"). Scopes may be nested. In different scopes, the same <correlation name> may be associated with different tables or with the same table.

21) No <authorization identifier> shall specify “PUBLIC”.

22) Those <identifier>s that are valid <authorization identifier>s are implementation-defined.

23) A <template parameter name> shall appear in a <type template definition>.

24) Any <template parameter name> contained in the <abstract data type body> of a <type template definition> shall also be contained in a <template parameter declaration> of the same <type template definition>.

Case:

a) A <template parameter name> that is contained in <data type> shall be contained in a <template parameter declaration> whose <template parameter type> is TYPE.

b) A <template parameter name> that is contained in <value specification> or <target specification> shall be contained in a <template parameter declaration> whose <template parameter type> is a <data type>.

25) Those <identifier>s that are valid <catalog name>s are implementation-defined.

26) If a <character set name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

27) If a <collation name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

28) If a <translation name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

29) The <data type> of <SQL-server name>, <connection name>, and <user name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

30) If a <form-of-use conversion name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit; otherwise, INFORMATION_SCHEMA shall be specified.

Access Rules

None.

General Rules

1) A <table name> identifies a table.

2) Within its scope, a <correlation name> identifies a table.

ISO Only–SQL3

3) Within its scope, a <query name> identifies the table defined or returned by some associated <query expression body>.

4) A <column name> identifies a column.

5) A <domain name> identifies a domain.

6) An <authorization identifier> represents an authorization identifier and identifies a set of privileges.

7) An <enumeration name> identifies a component of an <enumerated type>.

8) A <module name> identifies a <module>.

9) A <cursor name> identifies a cursor.

• 1 Rule moved to Part 4

10) A <parameter name> identifies a parameter.

11) An <external routine name> identifies an external routine.

12) A <trigger name> identifies a trigger.

13) A <constraint name> identifies a table constraint, a domain constraint, or an assertion.

14) A <catalog name> identifies a catalog.

15) A <schema name> identifies a schema.

16) A <collation name> identifies a collating sequence.

17) A <character set name> identifies a character set.

18) A <translation name> identifies a character translation.

19) A <form-of-use conversion name> identifies a form-of-use conversion. All <form-of-use conversion name>s are implementation-defined.

20) A <connection name> identifies an SQL-connection.

21) An <abstract data type name> identifies an abstract data type.

22) A <distinct type name> identifies a distinct type.

23) A <component name> identifies a component of an abstract data type.

24) An <attribute name> identifies an attribute of an abstract data type.

25) A <savepoint name> identifies a savepoint. The scope of a <savepoint name> is the SQL-transaction in which it was defined.

26) A <field name> identifies a field.

27) An <operator name> identifies an operator.

28) A <role name> identifies a role.

29) A <null class name> identifies a defined null class. No <null class name> shall specify GENERAL.

30) If the <form-of-use conversion name> does not contain an explicit a <schema name>, then INFORMATION_SCHEMA is implicit; otherwise, INFORMATION_SCHEMA shall be specified.

31) A <type template name> identifies a type template family.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <component name>.

   b) Conforming Full SQL language shall not contain any <abstract data type name>.

   c) Conforming Full SQL language shall not contain any <savepoint name>.

   d) Conforming Full SQL language shall not contain any <role name>.

   e) Conforming Full SQL language shall not contain any <null class name>.

   f) Conforming Full SQL language shall not contain any <external routine name>.

   ISO Only–SQL3

   g) Conforming Full SQL language shall not contain any <query name>.

   h) Conforming Full SQL language shall not contain any <operator name>.

   i) Conforming Full SQL language shall not contain any <type template name>.

   j) Conforming Full SQL language shall not contain any <trigger name>.

   k) Conforming Full SQL language shall not contain any <enumeration name>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any explicit <catalog name>, <connection name>, <collation name>, <translation name>, <form-of-use conversion name>, or <qualified local table name>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <domain name>, <constraint name>, or <character set name>.

   b) An <identifier> shall not specify a <character set specification>.

   c) A <parameter name> shall immediate contain a <colon>.

6 Scalar expressions

6.1 <data type>

Function

Specify a data type.

Format

```
<data type> ::=
        <predefined type>
      | <row type>
      | <user-defined type>
      | <template parameter name>
      | <collection type>

<predefined type> ::=
        <character string type> [ CHARACTER SET <character set specification> ]
      | <national character string type>
      | <binary large object string type>
      | <bit string type>
      | <numeric type>
      | <enumerated type>
      | <boolean type>
      | <datetime type>
      | <interval type>
```

ANSI Only--SQL3

```
      | <row identifier type>
```

```
<character string type> ::=
        CHARACTER [ <left paren> <length> <right paren> ]
      | CHAR [ <left paren> <length> <right paren> ]
      | CHARACTER VARYING <left paren> <length> <right paren>
      | CHAR VARYING <left paren> <length> <right paren>
      | VARCHAR <left paren> <length> <right paren>
      | CHARACTER LARGE OBJECT [ <left paren> <large object length> <right paren> ]
      | CHAR LARGE OBJECT [ <left paren> <large object length> <right paren> ]
      | CLOB [ <left paren> <large object length> <right paren> ]

<national character string type> ::=
        NATIONAL CHARACTER [ <left paren> <length> <right paren> ]
      | NATIONAL CHAR [ <left paren> <length> <right paren> ]
      | NCHAR [ <left paren> <length> <right paren> ]
      | NATIONAL CHARACTER VARYING <left paren> <length> <right paren>
      | NATIONAL CHAR VARYING <left paren> <length> <right paren>
      | NCHAR VARYING <left paren> <length> <right paren>
      | NATIONAL CHARACTER LARGE OBJECT [ <left paren> <large object length> <right paren> ]

      | NCHAR LARGE OBJECT [ <left paren> <large object length> <right paren> ]
      | NCLOB [ <left paren> <large object length> <right paren> ]
```

```
<binary large object string type> ::=
        BINARY LARGE OBJECT [ <left paren> <large object length> <right paren> ]
      | BLOB [ <left paren> <large object length> <right paren> ]

<bit string type> ::=
        BIT [ <left paren> <length> <right paren> ]
      | BIT VARYING <left paren> <length> <right paren>

<numeric type> ::=
        <exact numeric type>
      | <approximate numeric type>

<exact numeric type> ::=
        NUMERIC [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
      | DECIMAL [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
      | DEC [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
      | INTEGER
      | INT
      | SMALLINT

<approximate numeric type> ::=
        FLOAT [ <left paren> <precision> <right paren> ]
      | REAL
      | DOUBLE PRECISION

<length> ::= <unsigned integer>

<large object length> ::=
        <unsigned integer>
      | <unsigned integer> K
      | <unsigned integer> M
      | <unsigned integer> G

<precision> ::= <unsigned integer>

<scale> ::= <unsigned integer>

<enumerated type> ::= <left paren> <enumeration name list> <right paren>

<enumeration name list> ::=
      <enumeration name> [ { <comma> <enumeration name> }... ]

<boolean type> ::= BOOLEAN

<datetime type> ::=
        DATE
      | TIME [ <left paren> <time precision> <right paren> ] [ WITH TIME ZONE ]
      | TIMESTAMP [ <left paren> <timestamp precision> <right paren> ] [ WITH TIME ZONE ]


<time precision> ::= <time fractional seconds precision>

<timestamp precision> ::= <time fractional seconds precision>

<time fractional seconds precision> ::= <unsigned integer>

<interval type> ::= INTERVAL <interval qualifier>

<row type> ::=
      ROW <left paren>
```

```
        <field definition> [ { <comma> <field definition> }... ]
      <right paren>

<user-defined type> ::=
        <abstract data type>
      | <distinct type name>
      | <generated type reference>

<abstract data type> ::=
      <abstract data type name> [ <elaboration mode> ]

<elaboration mode> ::= INSTANCE

<collection type> ::=
```

ANSI Only--SQL3

```
        <general collection type>
      | <set type>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
        <set type>
```

```
      | <multiset type>
      | <list type>
```

ANSI Only--SQL3

```
<general collection type> ::=
      COLLECTION <left paren> <data type> <right paren>
```

```
<set type> ::=
      SET <left paren> <data type> <right paren>

<multiset type> ::=
      MULTISET <left paren> <data type> <right paren>

<list type> ::=
      LIST <left paren> <data type> <right paren>
```

ANSI Only--SQL3

```
<row identifier type> ::= <table name> IDENTITY
```

Syntax Rules

1) CHAR is equivalent to CHARACTER. DEC is equivalent to DECIMAL. INT is equivalent to INTEGER. VARCHAR is equivalent to CHARACTER VARYING. NCHAR is equivalent to NATIONAL CHARACTER. CLOB is equivalent to CHARACTER LARGE OBJECT. NCLOB is equivalent to NATIONAL CHARACTER LARGE OBJECT. BLOB is equivalent to BINARY LARGE OBJECT.

2) "NATIONAL CHARACTER" is equivalent to the corresponding <character string type> with a specification of "CHARACTER SET *CSN*", where "*CSN*" is an implementation-defined <character set name>.

3) The value of a <length> or a <precision> shall be greater than 0.

4) If <length> is omitted, then a <length> of 1 is implicit.

5) If <large object length> is omitted, then an implementation-defined <large object length> is implicit.

6) If <unsigned integer> K is specified, then it is equivalent to the <unsigned integer> resulting from the multiplication of the specified <unsigned integer> and 1024.

7) If <unsigned integer> M is specified, then it is equivalent to the <unsigned integer> resulting from the multiplication of the specified <unsigned integer> and 1,028,476.

8) If <unsigned integer> G is specified, then it is equivalent to the <unsigned integer> resulting from the multiplication of the specified <unsigned integer> and 1,073,640,424.

9) If a <scale> is omitted, then a <scale> of 0 is implicit.

10) If a <precision> is omitted, then an implementation-defined <precision> is implicit.

11) CHARACTER specifies the data type character string.

12) Characters in a character string are numbered beginning with 1.

13) Case:

    a) If neither VARYING nor LARGE OBJECT is specified in <character string type>, then the length in characters of the character string is fixed and is the value of <length>.

    b) If VARYING is specified in <character string type>, then the length in characters of the character string is variable, with a minimum length of 0 and a maximum length of the value of <length>.

    c) LARGE OBJECT is contained in a <character string type>, then the length in characters of the character string is variable, with a minimum length of 0 and a maximum length of the value of <large object length>.

    The maximum values of <length> and <large object length> are implementation-defined. Neither <length> nor <large object length> shall be greater than that maximum value.

14) If <character string type> is not contained in a <domain definition> or a <column definition> and CHARACTER SET is not specified, then an implementation-defined <character set specification> is implicit.

Note: Subclause 11.28, "<domain definition>", and Subclause 11.6, "<column definition>", specify the result when <character string type> is contained in a <domain definition> or <column definition>, respectively.

15) The character set named SQL_TEXT is an implementation-defined character set whose character repertoire is SQL_TEXT.

Note: The character repertoire SQL_TEXT is defined in Subclause 4.2, "Character strings".

16) BINARY LARGE OBJECT specifies the data type binary string.

17) Octets in a binary large object string are numbered begining with 1. The length in octets of the string is variable, with a minimum length of 0 and a maximum length of the value of <large object length>.

18) BIT specifies the data type bit string.

19) Bits in a bit string are numbered beginning with 1.

20) Case:

a) If VARYING is not specified in <bit string type>, then the length in bits of the bit string is fixed and is the value of <length>.

b) If VARYING is specified in <bit string type>, then the length in bits of the string is variable, with a minimum length of 0 and a maximum length of the value of <length>.

The maximum value of <length> is implementation-defined. <length> shall not be greater than this maximum value.

21) The <scale> of an <exact numeric type> shall not be greater than the <precision> of the <exact numeric type>.

22) For the <exact numeric type>s DECIMAL and NUMERIC:

a) The maximum value of <precision> is implementation-defined. <precision> shall not be greater than this value.

b) The maximum value of <scale> is implementation-defined. <scale> shall not be greater than this maximum value.

23) NUMERIC specifies the data type exact numeric, with the decimal precision and scale specified by the <precision> and <scale>.

24) DECIMAL specifies the data type exact numeric, with the decimal scale specified by the <scale> and the implementation-defined decimal precision equal to or greater than the value of the specified <precision>.

25) INTEGER specifies the data type exact numeric, with binary or decimal precision and scale of 0. The choice of binary versus decimal precision is implementation-defined, but shall be the same as SMALLINT.

26) SMALLINT specifies the data type exact numeric, with scale of 0 and binary or decimal precision. The choice of binary versus decimal precision is implementation-defined, but shall be the same as INTEGER. The precision of SMALLINT shall be less than or equal to the precision of INTEGER.

27) FLOAT specifies the data type approximate numeric, with binary precision equal to or greater than the value of the specified <precision>. The maximum value of <precision> is implementation-defined. <precision> shall not be greater than this value.

28) REAL specifies the data type approximate numeric, with implementation-defined precision.

29) DOUBLE PRECISION specifies the data type approximate numeric, with implementation-defined precision that is greater than the implementation-defined precision of REAL.

30) For the <approximate numeric type>s FLOAT, REAL, and DOUBLE PRECISION, the maximum and minimum values of the exponent are implementation-defined.

31) An <enumerated type> shall be contained in a <domain definition>.

32) In an <enumerated type>, each <enumeration name> shall be different from any other <enumeration name> of the <enumerated type>.

33) An <enumerated type> defines an ordering of <enumeration name>s. Each <enumeration name> yields a different enumeration value. The predefined order relation between enumeration values follows the order of corresponding position numbers. The position number of the value of the first listed <enumeration name> is 0. The position number of each other <enumeration name> is 1 more than the position number of its predecessor in the list.

34) If <time precision> is not specified, then 0 is implicit. If <timestamp precision> is not specified, then 6 is implicit.

35) The maximum value of <time precision> and the maximum value of <timestamp precision> shall be the same implementation-defined value that is not less than 6. The values of <time precision> and <timestamp precision> shall not be greater than that maximum value.

36) The length of a DATE is 10 positions. The length of a TIME is 8 positions plus the <time fractional seconds precision>, plus 1 position if the <time fractional seconds precision> is greater than 0. The length of a TIME WITH TIME ZONE is 14 positions plus the <time fractional seconds precision> plus 1 position if the <time fractional seconds precision> is greater than 0. The length of a TIMESTAMP is 19 positions plus the <time fractional seconds precision>, plus 1 position if the <time fractional seconds precision> is greater than 0. The length of a TIMESTAMP WITH TIME ZONE is 25 positions plus the <time fractional seconds precision> plus 1 position if the <time fractional seconds precision> is greater than 0.

37) If an <interval qualifier> in an <interval type> includes no fields other than YEAR and MONTH, then the <interval type> is a year-month interval. If an <interval qualifier> in an <interval type> includes any fields other than YEAR or MONTH, then the <interval type> is a day-time interval.

38) The *i*-th value of an interval data type corresponds to the *i*-th <datetime field>.

39) Within the non-null values of a <datetime type>, the value of the time zone interval shall be in the range −12:59 to +13:00.

Note: The range for time zone intervals is larger than many readers might expect because it is governed by political decisions in governmental bodies rather than by any natural law.

40) If <data type> simply contains an <abstract data type> *ADT*, then:

a) There shall exist an abstract data type descriptor *ADTD* specifying the <abstract data type name> of *ADT*.

b) If *ADT* specifies INSTANCE, then *ADTD* shall be an object *ADT*.

41) If <data type> immediately contains a <distinct type name>, then there shall exist a distinct type descriptor whose distinct type name is <distinct type name>.

ANSI Only–SQL3

42) <general collection type> specifies the general collection data type.

43) <set type> specifies the set data type.

44) <multiset type> specifies the multiset data type.

45) <list type> specifies the list data type.

ANSI Only–SQL3

46) The row type of a <collection type> is described by a single (<field name>, data type) pair, where the data type is that specified by the <data type> simply contained in <collection type> and the <field name> is implementation-dependent and different from the name of any column or field, other than itself, of a table referenced by any <table reference> contained in the SQL-statement.

47) Let *T* be the table identified in <row identifier type>. *T* shall have an implicit row identifier column.

48) <row type> specifies the row data type.

Access Rules

1) The applicable privileges shall include the USAGE privilege on the abstract data type identified by <abstract data type name>.

   Note: The applicable privileges for an <abstract data type name> are defined in Subclause 10.4, "<privileges>".

2) The applicable privileges shall include the USAGE privilege on the distinct type identified by <distinct type name>.

   Note: The applicable privileges for an <abstract data type name> are defined in Subclause 10.4, "<privileges>".

General Rules

1) If any specification or operation attempts to cause an item of a character type to contain a character that is not a member of the character repertoire associated with the character item, then an exception condition is raised: *data exception—character not in repertoire*.

2) For a <datetime type>,

Case:

a) If DATE is specified, then the data type contains the <datetime field>s years, months, and days.

b) If TIME is specified, then the data type contains the <datetime field>s hours, minutes, and seconds.

c) If TIMESTAMP is specified, then the data type contains the <datetime field>s years, months, days, hours, minutes, and seconds.

3) For a <datetime type>, a <time fractional seconds precision> that is an explicit or implicit <time precision> or <timestamp precision> defines the number of decimal digits following the decimal point in the SECOND <datetime field>.

4) Table 11, "Valid values for fields in datetime items", speifies the constraints on the values of the <datetime field>s in a datetime data type.

Table 11—Valid values for fields in datetime items

Keyword	Valid values of datetime fields
YEAR	0001 to 9999
MONTH	01 to 12
DAY	Within the range 1 to 31, but further constrained by the value of MONTH and YEAR fields, according to the rules for well-formed dates in the Gregorian calendar.
HOUR	00 to 23
MINUTE	00 to 59
SECOND	00 to 61.9(*N*) where "9(*N*)" indicates the number of digits specified by <time fractional seconds precision>.
TIMEZONE_HOUR	00 to 13
TIMEZONE_MINUTE	00 to 59

Note: Datetime data types will allow dates in the Gregorian format to be stored in the date range 0001−01−01 CE through 9999−12−31 CE. The range for SECOND allows for as many as two "leap seconds". Interval arithmetic that involves leap seconds or discontinuities in calendars will produce implementation-defined results.

5) If WITH TIME ZONE is not specified, then the time zone displacement of the datetime data type is effectively the current default time zone displacement of the SQL-session.

6) The values of the <datetime field>s within an interval data type are constrained as follows:

a) The value corresponding to the first <datetime field> is an integer with at most *N* digits, where *N* is the <interval leading field precision>.

b) Table 12, "Valid values for fields in INTERVAL items", specifies the constraints for the other <datetime field>s in the interval data type.

Table 12—Valid values for fields in INTERVAL items

Keyword	Valid values of INTERVAL fields
MONTH	0 to 11
HOUR	0 to 23
MINUTE	0 to 59
SECOND	0 to 59.9(*N*) where "9(*N*)" indicates the number of digits specified by <interval fractional seconds precision> in the <interval qualifier>.

7) An item of type interval can contain positive or negative intervals.

8) If a <data type> is an <abstract data type>, then

   Case:

   a) If the <abstract data type> is an object ADT and the <data type> does not specify INSTANCE, then the item contains an object identifier that identifies an instance of the abstract data type.

   b) Otherwise, the item contains an instance of the abstract data type.

ANSI Only–SQL3

9) For any <data type> *DT*, the type SET(*DT*) is a subtype of the type COLLECTION(*DT*).

10) For any <data type> *DT*, the type MULTISET(*DT*) is a subtype of the type COLLECTION(*DT*).

11) For any <data type> *DT*, the type LIST(*DT*) is a subtype of the type COLLECTION(*DT*).

12) If the <data type> is a <collection type>, then a collection type descriptor is created. The collection type descriptor includes the name of the data type of elements and an indication of whether the collection is a set, multiset, or list.

13) For a <row type> *RT*, the degree of *RT* is initially set to zero. The General Rules of Subclause 11.8, "<field definition>", specify the degree of *RT* during the definition of the fields of *RT*.

14) If the <data type> is a <row type>, then a row type descriptor is created. The row type descriptor includes the degree of the row type and a field descriptor for every <field definition> of the <row type>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <data type> shall not be a <user-defined type>,
   ANSI <template parameter name>, or <row identifier type>.
   ISO or <template parameter name>.

b) A <predefined type> shall not be an <enumerated type> or <boolean type>.

c) A <data type> shall not be a <collection type>.

d) A <data type> shall not be a <binary large object string type>.

e) A <character string type> shall not specify CHARACTER LARGE OBJECT, CHAR LARGE OBJECT, or CLOB.

f) A <national character string type> shall not specify NATIONAL CHARACTER LARGE OBJECT, NCHAR LARGE OBJECT, or NCLOB.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <datetime type> shall not specify a <time precision> or <timestamp precision>.

b) A <data type> shall not be a <bit string type>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <character string type> shall not specify VARYING or VARCHAR.

b) A <data type> shall not be a <datetime type> or an <interval type>.

c) A <data type> shall not be a <national character string type> nor specify CHARACTER SET.

6.2 <value specification> and <target specification>

Function

Specify one or more values, parameters, or variables.

Format

```
<value specification> ::=
        <literal>
      | <general value specification>

<unsigned value specification> ::=
        <unsigned literal>
      | <general value specification>

<general value specification> ::=
        <item reference>
      | USER
      | CURRENT_USER
      | SESSION_USER
      | SYSTEM_USER
      | CURRENT_PATH
      | VALUE
```

• 1 alternative moved to Part 4

```
      | <field reference>
      | <component reference>
      | <template parameter name>

<simple value specification> ::=
        <item reference>
      | <literal>

<target specification> ::=
        <item reference>
      | <template parameter name>

<simple target specification> ::=
      <item reference>
```

Syntax Rules

1) An <item reference> that is a <general value specification> or <target specification> shall be a parameter reference, an SQL variable reference, or an <embedded variable specification>.

   Note: "Parameter reference" and "SQL variable reference" are defined in Subclause 6.3, "<item reference>".

2) An <item reference> that is a <simple value specification> or <simple target specification> shall be a host parameter reference.

   Note: "Host parameter reference" is defined in Subclause 6.3, "<item reference>".

3) If USER is specified, then CURRENT_USER is implicit.

Note: In an environment where the SQL-implementation conforms to Entry SQL, conforming SQL language that contains either:

a) a specified or implied <comparison predicate> that compares the <value specification> USER with a <value specification> other than USER, or

b) a specified or implied assignment in which the "value" (as defined in Subclause 9.2, "Store assignment") contains the <value specification> USER

will become non-conforming in an environment where the SQL-implementation conforms to Intermediate SQL or Full SQL, unless the character repertoire of the implementation-defined character set in that environment is identical to the character repertoire of SQL_TEXT.

4) The data type of CURRENT_USER, SESSION_USER, SYSTEM_USER, and CURRENT_PATH is character string. Whether the character string is fixed length or variable length, and its length if it is fixed length or maximum length if it is variable length, are implementation-defined. The character set of the character string is SQL_TEXT. The null class of the character string is the general null class.

ANSI Only–SQL3

---

5) The <value specification> or <unsigned value specification> VALUE shall be contained either in a <domain constraint> or in an <abstract data type body>.

Case:

a) If VALUE is contained in a <domain constraint>, then the data type of an instance of VALUE is the data type of the domain to which that domain constraint belongs. The null class of an instance of VALUE is the null class of the domain definition to which that domain constraint belongs.

b) If VALUE is contained in an <abstract data type body> *ADTB*, then the data type of an instance of VALUE is

Case:

i) If *ADTB* is contained in an <abstract data type definition>, then the abstract data type of which *ADTB* is the body; or

ii) If *ADTB* is contained in a <type template definition>, then :GEN_TYPE.

ISO Only—caused by ANSI changes not yet considered by ISO

---

6) The <value specification> or <unsigned value specification> VALUE shall be contained in a <domain constraint>. The data type of an instance of VALUE is the data type of the domain to which that domain constraint belongs. The null class of an instance of VALUE is the null class of the <domain definition> containing that <domain constraint>.

---

7) If the data type of the <value specification> or <unsigned value specification> is character string, then the <value specification> or <unsigned value specification> has the *Coercible* coercibility attribute, and the collating sequence is determined by Subclause 4.2.3, "Rules determining collating sequence usage".

8) The null class of a parameter reference is the general null class.

- 1 Rule moved to Part 4

Access Rules

None.

General Rules

1) A <value specification> or <unsigned value specification> specifies a value that is not selected from a table.

2) A <target specification> specifies a parameter or variable that can be assigned a value.

3) The value specified by a <literal> is the value represented by that <literal>.

4) The value specified by CURRENT_USER is the value of the current <authorization identifier>.

5) The value specified by SESSION_USER is the value of the SQL-session <authorization identifier>.

6) The value specified by SYSTEM_USER is equal to an implementation-defined string that represents the operating system user who executed the <module> that contains the SQL-statement whose execution caused the SYSTEM_USER <general value specification> to be evaluated.

7) The value specified by CURRENT_PATH is a <schema name list> where <catalog name>s are <delimited identifier>s and the <unqualified schema name>s are <delimited identifier>s. Each <schema name> is separated from the preceding <schema name> by a <comma> with no intervening <space>s.

8) A <simple value specification> specifies a <value specification> or <unsigned value specification> that is not null and does not have an associated <indicator parameter>.

9) A <simple target specification> specifies a parameter or variable that can be assigned a value that is not null.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <general value specification> shall not be a <template parameter name>, <function name>, <item reference>, <component reference>, or CURRENT_PATH.

   b) A <simple value specification> shall not be an an <item reference>.

   c) A <target specification> shall not be a <template parameter name> or <component reference>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <general value specification> shall not specify VALUE.

   b) A <general value specification> shall not specify CURRENT_USER, SYSTEM_USER, or SESSION_USER.

      Note: Although CURRENT_USER and USER are semantically the same, in Entry SQL, CURRENT_USER must be specified as USER.

6.3 <item reference>

Function

Reference a column, parameter, or variable.

Format

```
<item reference> ::=
       [ <item qualifier> <period> ]<item name> [ <indicator parameter> ]

<item qualifier> ::=
         <table name>
       | <correlation name>
       | <routine name>

<item name> ::=
         <column name>
       | <parameter name>
```

ANSI Only--SQL3

```
       | IDENTITY
```

```
<indicator parameter> ::= [ INDICATOR ] <parameter name>
```

Syntax Rules

1) Let *IR* be an <item reference> and let *IN* be the <item name> contained in *IR*.

2) If *IN* is a <parameter name> that simply contains a <colon>, then *IR* shall not contain an <item qualifier>.

ANSI Only–SQL3

3) If *IN* is "IDENTITY", then *IR* shall contain neither an <item qualifier> that is a <routine name> nor an <indicator parameter>.

4) If *IR* contains an <item qualifier> *IQ*, then *IR* shall appear within the scope of one or more exposed
ANSI <table name>s,
ISO <table or query name>s,
<correlation name>s, or <routine>s that are equal to *IQ*. If there is more than one such exposed

ANSI <table name>,
ISO <table or query name>,

<correlation name>, or <routine name>, then the one with the most local scope is specified. Let *V* be the table or parameter list associated with *IQ*.

a) *V* shall include a column or parameter whose <column name> or <parameter name> is *IN*.

b) If *V* is a <table reference> in a <joined table> *JT*, then *IN* shall not be a common column name in *JT*.

Note: "Common column name" is defined in Subclause 7.9, "<joined table>".

5) If *IR* does not contain an <item qualifier>, then *IR* shall be contained within the scope of one or more exposed
ANSI <table name>s,
ISO <table or query name>s,
<correlation name>s, or <routine>s whose associated tables or <parameter list>s include a column or parameter whose <identifier> is

ANSI Only–SQL3

IN, or where no table in the same scope has a column whose <column name> is *CN*, whose exposed
ANSI <table name>
ISO <table or query name>
or <correlation name> is *CN*.

ISO Only—caused by ANSI changes not yet considered by ISO

IN.

Let the phrase *possible qualifiers* denote those exposed
ANSI <table name>s,
ISO <table or query name>s,
<correlation name>s, and <routine name>s.

a) Case:

i) If the most local scope contains exactly one possible qualifier, then the qualifier *IQ* equivalent to that unique exposed
ANSI <table name>,
ISO <table or query name>,
<correlation name>, or <routine name> is impliict.

ANSI Only–SQL3

If *CN* is not a <column name> of *V*, then *C* identifies the implicit row identifier column of *V*.

ii) If there is more than one possible qualifier with the most local scope, then:

1) Each possible qualifier shall be a
ANSI <table name>
ISO <table or query name>

or a <correlation name> of a <table reference> that is direction contained in a <joined table> *JT*.

2) *CN* shall be a common column name in *JT*.

Note: "Common column name" is defined in Subclause 7.9, "<joined table>".

3) The implicit qualifier *IQ* is implementation-dependent. The scope of *IQ* is that which *IQ* would have had if *JT* had been replaced by the <table reference>:

(*JT*) AS *IQ*

b) Let *V* be the table or parameter list associated with *IQ*.

6) Case:

a) If *V* is a
ANSI <table name>
ISO <table or query name>
or <correlation name>, then *IR* is a *column reference*. *IN* shall uniquely identify a column of *V*. Let *R* be that column.

i) *R* is an underlying column of *IR*. If *R* is a <derived column>, then every underlying column of *R* is an underlying column of *IR*.

Note: The underlying columns of a <derived column> are defined in Subclause 7.13, "<query specification>".

ii) If *IR* is contained in a <table expression> *TE*
ANSI or a <there is predicate> *TE*
or a <quantified predicate>*TE*
ANSI immediately containing an <existential clause> or <universal clause>
and the scope of *IQ* is some <SQL procedure statement>, <recursive union>, <trigger definition>, <quantified predicate>, or <table reference> that contains *TE*, then *CR* is an *outer reference* to the table associated with *Q*.

b) If *V* is a <routine name>, then *IR* is a *parameter reference*. Let *R* be the parameter of *V* identified by *IN*. *IR* is a *host parameter reference*.

****Editor's Note****
Paper X3H2-94-244/SOU-097 noted that *V* is not a name of any sort, but is a table or parameter list. See Possible Problem **406** in the Editor's Notes.

7) An <item name> that is a <parameter name> shall simply contain a <colon>.

8) The <parameter name> of an <indicator parameter> shall identify a parameter of *V*. The data type of that parameter shall be exact numeric with a scale of 0.

9) If the data type of *R* is character string, then *R* has the *Implicit* coercibility attribute and its collating sequence is the default collating sequence for the column or parameter *R*.

10) If the data type fo *R* is TIME or TIMESTAMP, then the implicit time zone of the data is the current default time zone for the SQL-session.

11)

12) If the data type fo *R* is TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE, then the time zone of the data is the time zone represented in the value of *IR*.

13) Case:

a) If *IR* is contained in a <declare cursor>, then let *OS* be the <open statement>s in the containing <module> that specify the <cursor name> of the <declare cursor>. For each <open statement> *O* in *OS*, *O* shall be contained in a <routine> that contains a <parameter declaration> whose <parameter name> is *IR*; *IR* denotes that parameter.

b) Otherwise, *IR* shall be contained in one or more <routine>s that contain an <parameter declaration> whose <parameter name> is *IR*; *IR* denotes that parameter in the innermost such <routine>.

****Editor's Note****

The preceding Rule representes the Editor's best effort to merge the effects of papers X3H2-93-096 /MUN-069R and X3H2-93-140/YOK-101/MUN-067. It seems quite likely that additional work will be required to rationalize the intents of those two papers. See Possible Problem **320** in the Editor's Notes.

Access Rules

1) If *IR* is a column reference, then the applicable privileges shall include SELECT for *V* if *CR* is contained in any of:

a) a <query expression> simply contained in a <cursor specification>, a <view definition>, an <insert statement>, a <temporary view declaration>; or

b) a <sort specification list> contained in a <cursor specification>; or

c) a <table expression>
ANSI or <there is predicate>
immediately contained in a <select statement: single row>; or

d) a <search condition> immediately contained in a <delete statement: searched> or an <update statement: searched>; or

e) a <select list> immediately contained in a <select statement: single row>; or

f) a <value expression> immediately contained in an <update source>.

General Rules

1) Depending on whether *IR* is a column reference or parameter reference, *IQ.IN* references column *IN* in a given row of *V* or parameter *IN* of a given call of *V*.

2) If a host parameter reference contains an <indicator parameter> and the value of the indicator parameter is negative, then the value specified by the host parameter reference is a null value. Otherwise, the value specified by a host parameter reference is the value of the parameter.

3) If the data type of *IR* is TIME, TIMESTAMP, TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE, then let *TZ* be an INTERVAL HOUR TO MINUTE containing the value of the time zone displacement associated with *IR*. The value of *IR*, normalized to UTC, is effectively computed as:

$CR + TZ$

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <item reference> that contains an <item qualifier> shall be a column reference.

   ANSI Only–SQL3

   b) The <item reference> shall not identify an implicit row identifier column.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.4 <row reference>

Function

Reference a row.

Format

```
<row reference> ::= ROW <item qualifier>
```

Syntax Rules

1) A <row reference> *RR* with a <item qualifier> *Q* shall appear within the scope of one or more <table name>s or <correlation name>s that are equal to *Q*. If there is more than one such <table name> or <correlation name>, then the one with the most local scope is specified. Let *T* be the table associated with *Q*. The data type of *RR* is the row type of *Q*.

2) If *Q* is contained in a <table expression> *TE*
ANSI or a <there is predicate> *TE*
or a <quantified predicate> *TE*
ANSI immediately containing an <existential clause> or a <universal clause>
and the scope clause of *Q* is some <SQL procedure statement>, <recursive union>, <trigger definition>, <quantified predicate>, or <table reference> that contains *TE*, then *Q* is an *outer reference* to the table associated with *Q*.

Access Rules

None.

General Rules

1) A <row reference> *RR* references a given row *R* of *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

6.5 <component reference>

Function

References a function or an attribute in an abstract data type instance.

Format

```
<component reference> ::=
      <value specification> <double period> <component name>
```

Syntax Rules

1) The data type of <value specification> shall be an abstract data type. Let *ADT* be the abstract data type.

2) If <component reference> is not immediately contained in <general value specification>, then the <component name> shall not be OID.

Access Rules

None.

General Rules

1) Let *VS* be the value of <value specification>.

2) If *VS* is a null value, then an exception condition is raised: *data exception—null instance in attribute reference.*

3) Case:

   a) If the data type of <value specification> is an abstract data type WITHOUT OID, then let *I* denote the abstract data type instance identified by *VS*.

   b) If the data type of <value specification> is an object identifier type, then let *I* denote the abstract data type instance referenced by *VS*.

4) Let *I* be the abstract data type instance identified by the <value specification>. Let *F* be the <attribute name>.

5) Case:

   a) If <component reference> is immediately contained in <general value specification>, then the result of <component reference> is the result of the SQL <routine invocation> *F*(*I*).

   b) Otherwise, let *V* be the *VALUE* specified in an application of this Subclause. An SQL <routine invocation> of *F*(*I*,*V*) is performed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <component reference>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.6 <field reference>

Function

Access a field of a row type instance.

Format

```
<field reference> ::=
      <value expression> <double period> <field name>
```

Syntax Rules

1) The data type of <value expression> shall be a row type *RT*.

2) The <field name> *FN* shall be the name of a field of *RT*.

3) The data type of the <field reference> is the data type of the field of *RT* whose name is *FN*.

Access Rules

None.

General Rules

1) Let *VR* be the value of the <value expression>.

2) If *VR* is a null value, then an exception condition is raised: *data exception—null instance in field reference*.

3) The value of the <field reference> is the value of the field of *VR* whose name is *FN*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <field reference>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.7 <table reference>

Function

Reference a table.

Format

```
<table reference> ::=
```

ANSI Only--SQL3

```
        <table specification> [ [ AS ] <correlation name>
```

ISO Only--SQL3

```
        <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> ]
      | <joined table>

<derived table> ::= <table subquery>
```

ISO Only--SQL3

```
<table or query name> ::=
        <table name>
      | <query name>
```

```
<derived column list> ::= <column name list>

<column name list> ::=
      <column name> [ { <comma> <column name> }... ]
```

ANSI Only--SQL3

```
<table specification> ::=
        <table name>
      | <multi-table specification>
      | ONLY <table name>

<multi-table specification> ::=
      ALL <table name> [ <except table specification> ]

<except table specification> ::=
      <left paren> EXCEPT <table specification>
        [ { <comma> <table specification> }... ]
```

****Editor's Note****
Paper X3H2-94-092/SOU-072 noted that the Syntax, Access, and General Rules in this Subclause do not take into account all of the possibilities in the Format. See Possible Problem **412** in the Editor's Notes.

Syntax Rules

1) A <correlation name> immediately contained in a <table reference> *TR* is *exposed* by *TR*. A
ANSI <table name>
ISO <table or query name>
immediately contained in a <table reference> *TR* is *exposed* by *TR* if and only if *TR* does not specify a <correlation name>.

2) Case:

   a) If a <table reference> *TR* is contained in a <from clause>
   ANSI or <there is clause>
   FC with no intervening <derived table>, then the *scope clause SC* of *TR* is the <select statement: single row> or innermost <query specification> that contains *FC*. The scope of the exposed <correlation name> or exposed
   ANSI <table name>
   ISO <table or query name>
   of *TR* is the <select list>, <where clause>, <group by clause>, and <having clause> of *SC*, together with the <join condition> of all <joined table>s contained in *SC* that contains *TR*.

   b) If a <table reference> *TR* is contained in an <existential clause> or <universal clause> *FC* with no intervening <derived table>, then the scope clause *SC* of *TR* is the innermost <quantified predicate> that contains *FC*. The scope of the exposed <correlation name> or exposed
   ANSI <table name>
   ISO <table or query name>
   of *TR* is *SC*, together with the <join condition> of all <joined table>s contained in *SC* that contain *TR*.

   c) Otherwise, the *scope clause SC* of *TR* is the outermost <joined table> that contains *TR* with no intervening <derived table>. The scope of the exposed <correlation name> or exposed
   ANSI <table name>
   ISO <table or query name>
   of *TR* is the <join condition> of *SC* and of all <joined table>s contained in *SC* that contain *TR*.

3) A
ANSI <table name>
ISO <table or query name>
that is exposed by a <table reference> *TR* shall not be the same as any other
ANSI <table name>
ISO <table or query name>
that is exposed by a <table reference> with the same scope clause as *TR*.

4) A <correlation name> that is exposed by a <table reference> *TR* shall not be the same as any other <correlation name> that is exposed by a <table reference> with the same scope clause as *TR* and shall not be the same as the <qualified identifier> of any
ANSI <table name>

ISO <table or query name>
that is exposed by a <table reference> with the same scope clause as *TR*.

5) A
ANSI <table name>
ISO <table or query name>
immediately contained in a <table reference> *TR* has a scope clause and scope defined by that <table reference> if and only if the
ANSI <table name>
ISO <table or query name>
is exposed by *TR*.

6) The same <column name> shall not be specified more than once in a <derived column list>.

7) If a <derived column list> is specified in a <table reference>, 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> or the
ANSI <table name>
ISO <table or query name>
of that <table reference>, and the name of the *i*-th column of that <derived table> or the effective name of the *i*-th column of that
ANSI <table name>
ISO <table or query name>
is the *i*-th <column name> in that <derived column list>.

8) Case:

   a) If no <derived column list> is specified, then the row type of the <table reference> is the row type of its immediately contained <table name>, <derived table>, or <joined table>.

   b) Otherwise, the row type of the <table reference> is described by a sequence of (<field name>, data type) pairs, where the <field name> in the *i*-th pair is the *i*-th <column name> in the <derived column list> and the data type in the *i*-th pair is the data type of the *i*-th column of the <derived table> or of the table identified by the <table name> immediately contained in the <table reference>.

ANSI Only–SQL3

9) Case:

   a) If no <derived column list> is specified, then the row type of the <table reference> is the row type of its immediately contained
   ANSI <table name>,
   ISO <table or query name>,
   <derived table>, or <joined table>.

   b) Otherwise, the row type of the <table reference> is described by a sequence of (<column name>, data type) pairs, where the <column 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 data type of the *i*-th column of the <derived table> or of the table identified by the
   ANSI <table name>
   ISO <table or query name>
   immediately contained in the <table reference>.

10) A <derived table> is an *inherently updatable table* if and only if the <query expression> simply contained in the <subquery> of the <table subquery> of the <derived table> is inherently updatable.

ISO Only—caused by ANSI changes not yet considered by ISO

11) A <derived table> is an *inherently updatable derived table* if and only if the <query expression> simply contained in the <subquery> of the <table subquery> of the <derived table> is inherently updatable.

ANSI Only–SQL3

12) Let *T* be the table identified by the <table name> immediately contained in <table reference>. If the <table reference> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*. If the <table reference> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T* or *S* shall include a <schema element> that creates the descriptor of *T*.

ISO Only–SQL3

13) If the <table or query name> immediately contained in <table reference> is not a *query name in scope*, then let *T* be the table identified by the <table name> immediately contained in <table or query name>. If the <table reference> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*. If the <table reference> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T* or *S* shall include a <schema element> that creates the descriptor of *T*.

Note: "*query name in scope*" is defined in Subclause 7.14, "<query expression>".

ANSI Only–SQL3

14) Every
ANSI <table name>
ISO <table or query name>
contained in an <except table specification> *ETS* shall identify a table in the subtable family of *T* (including *T*).

Access Rules

1) Let *T* be the table identified by the <table name> immediately contained in
ISO the <table or query name> immediately contained in
<table reference>.

2) If the <table reference> is contained in any of:

   a) a <query expression> simply contained in a <cursor specification>, a <view definition>, a <temporary view declaration>, or an <insert statement>; or

   b) a <table expression> or <select list> immediately contained in a <select statement: single row>; or

   c) a <search condition> immediately contained in a <delete statement: searched> or an <update statement: searched>; or

   d) a <value expression> immediately contained in an <update source>,

   then the applicable privileges shall include SELECT for
   ANSI *T* or for
   at least one column of *T*.

3) If the <table reference> is contained in a <query expression> simply contained in a <view definition> then the user privileges of the current <authorization identifier> shall include SELECT for at least one column of *T*.

General Rules

1) The <correlation name> or exposed
ANSI <table name>
ISO <table or query name>
contained in a <table reference> defines that <correlation name> or
ANSI <table name>
ISO <table or query name>
to be an identifier of the table identified by the
ANSI <table name>
ISO <table or query name>
or <derived table> of that <table reference>.

ANSI Only–SQL3

Let *T* be the table identified by
ANSI <table name>
ISO <table or query name>
contained in a <table specification> *TS*.

Case:

a) If ALL is specified and EXCEPT is not specified, then *TS* identifies *T*.

b) If both ALL and EXCEPT are specified, then *TS* identifies a table of the rows of *T* that have no corresponding rows in the table identified by any <table specification> immediately contained in *ETS*.

Note: The rules for table inheritance enforce that all rows of *T* be uniquely identifiable.

c) If ONLY is specified, then *TS* identifies a table fo the rows that do not have any corresponding row in any subtable of *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

ANSI Only–SQL3

a) A <table specification> shall be a <table name>.

ISO Only—caused by ANSI changes not yet considered by ISO

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <table reference> shall not be a <derived table>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <table reference> shall not be a <joined table>.

b) The optional <key word> AS shall not be specified.

c) <derived column list> shall not be specified.

6.8 <set function specification>

Function

Specify a value derived by the application of a function to an argument.

Format

```
<set function specification> ::=
        COUNT <left paren> <asterisk> <right paren>
      | <general set function>

<general set function> ::=
        <set function type>
            <left paren> [ <set quantifier> ] <value expression> <right paren> ]

<set function type> ::=
      AVG | MAX | MIN | SUM | COUNT

<set quantifier> ::= DISTINCT | ALL
```

Syntax Rules

1) If <set quantifier> is not specified, then ALL is implicit.

2) The argument of COUNT(*) and the argument source of a <general set function> is a table or a group of a grouped table as <general set function> is a table or a group of a grouped table as specified in Subclause 7.12, "<having clause>", and Subclause 7.13, "<query specification>".

3) Let *T* be the argument or argument source of a <set function specification>.

4) The <value expression> simply contained in <set function specification> shall not contain a <set function specification> or a <subquery>. If the <value expression> contains a column reference that is an outer reference, then that outer reference shall be the only column reference contained in the <value expression>.

   Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

5) If a <set function specification> contains a column reference that is an outer reference, then the <set function specification> shall be contained in either:

   a) a <select list>, or

   b) a <subquery> of a <having clause>, in which case the scope of the explicit or implicit <item qualifier> of the column reference shall be a <table reference> that is directly contained in the <table expression> that directly contains the <having clause>.

   Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

6) If the <set function specification> specifies a <set function type> that is MAX or MIN and the <value expression> contained in a <set function specification> simply contains a <value specification> whose data type is an abstract data type, then the <abstract data type definition> that defined that abstract data type shall contain a <less-than clause> that does not specify NONE.

7) If the <set function specification> specifies a <set function type> that is AVG, MAX, MIN, or SUM, then the data type of <value expression> simply contained in <general set function> shall not be a <collection type>.

8) The data type of the <value expression> simply contained in <general set function> shall not be a large object string.

9) If the <set function specification> specifies a <set function type> that is AVG or SUM, then the <value expression> contained in the <set function specification> shall not simply contain a <value specification> whose data type is an abstract data type.

10) Let *DT* be the data type of the <value expression>.

11) The data type of *DT* shall not be a distinct type.

12) If COUNT is specified, then the data type of the result is exact numeric with implementation-defined precision and scale of 0.

13) If MAX or MIN is specified, then the data type of the result is *DT*.

14) If SUM or AVG is specified, then:

    a) *DT* shall not be character string, bit string, an enumerated type, or datetime.

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

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

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

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

15) If the data type of the result is character string, then the collating sequence and the coercibility attribute are determined as in Subclause 4.2.3, "Rules determining collating sequence usage".

16) The null class of the result is the null class of the values that result from evaluation of <value expression>

Access Rules

None.

General Rules

1) Case:

    a) If COUNT(*) is specified, then the result is the cardinality of *T*.

    b) Otherwise, let *TX* be the single-column table that is the result of applying the <value expression> to each row of *T* and eliminating null values. If one or more null values are eliminated, then a completion condition is raised: *warning—null value eliminated in set function*.

2) If DISTINCT is specified, then let *TXA* be the result of eliminating redundant duplicate values from *TX*, using the comparison rules specified in Subclause 8.2, "<comparison predicate>" to identify the redundant duplicate values. Otherwise, let *TXA* be *TX*.

   Case:

   a) If the <general set function> COUNT is specified, then the result is the cardinality of *TXA*.

   b) If AVG, MAX, MIN, or SUM is specified, then

      Case:

      i) If *TXA* is empty, then the result is the general null value.

      ii) If AVG is specified, then the result is the average of the values in *TXA*.

      iii) If MAX or MIN is specified, then the result is respectively the maximum or minimum value in *TXA*. These results are determined using the comparison rules specified in Subclause 8.2, "<comparison predicate>".

      iv) If SUM is specified, then the result is the sum of the values in *TXA*. If the sum is not within the range of the data type of the result, then an exception condition is raised: *data exception—numeric value out of range*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) If a <general set function> specifies DISTINCT, then the <value expression> shall be a column reference.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If a <general set function> specifies or implies ALL, then COUNT shall not be specified.

   b) If a <general set function> specifies or implies ALL, then the <value expression> shall include a column reference that references a column of *T*.

   c) If the <value expression> contains a column reference that is an outer reference, then the <value expression> shall be a column reference.

   d) No column reference contained in a <set function specification> shall reference a column derived from a <value expression> that generally contains a <set function specification>.

6.9 <numeric value function>

Function

Specify a function yielding a value of type numeric.

Format

```
<numeric value function> ::=
         <position expression>
       | <extract expression>
       | <length expression>

<position expression> ::=
         <char position expression>
       | <blob position expression>

<char position expression> ::=
       POSITION <left paren> <string value expression>
           IN <string value expression> <right paren>

<blob position expression> ::=
       POSITION <left paren> <blob value expression>
       IN <blob value expression> <right paren>

<length expression> ::=
         <char length expression>
       | <octet length expression>
       | <bit length expression>

<char length expression> ::=
       { CHAR_LENGTH | CHARACTER_LENGTH }
           <left paren> <string value expression> <right paren>

<octet length expression> ::=
       OCTET_LENGTH <left paren> <string value expression> <right paren>

<bit length expression> ::=
       BIT_LENGTH <left paren> <string value expression> <right paren>

<extract expression> ::=
       EXTRACT <left paren> <extract field>
           FROM <extract source> <right paren>

<extract field> ::=
         <datetime field>
       | <time zone field>

<time zone field> ::=
         TIMEZONE_HOUR
       | TIMEZONE_MINUTE

<extract source> ::=
         <datetime value expression>
       | <interval value expression>
```

Syntax Rules

1) If <char position expression> is specified, then both <string value expression>s shall be <bit value expression>s or both shall be <character value expression>s having the same character repertoire.

2) If <position expression> is specified, then the data type of the result is exact numeric with implementation-defined precision and scale 0.

3) If <extract expression> is specified, then

   Case:

   a) If <extract field> is a <datetime field>, then it shall identify a <datetime field> of the <interval value expression> or <datetime value expression> immediately contained in <extract source>.

   b) If <extract field> is a <time zone field>, then the data type of the <extract source> shall be TIME WITH TIME ZONE or TIMESTAMP WITH TIME ZONE.

4) If <extract expression> is specified, then

   Case:

   a) If <datetime field> does not specify SECOND, then the data type of the result is exact numeric with implementation-defined precision and scale 0.

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

5) If a <length expression> is specified, then the data type of the result is exact numeric with implementation-defined precision and scale 0.

6) If all of the <character value expression>s, <string value expression>s, <datetime value expression>s, <interval value expression>s, and <blob value expression>s that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If the value of one or more <character value expression>s, <string value expression>s, <datetime value expression>s, and <interval value expression>s that are simply contained in a <numeric value function> are null values, then

   Case:

   a) If the result has the general null class, then the result of the <numeric value function> is the general null value.

b) Otherwise the result of the <numeric value function> is the minimum of the null values of the operands.

2) If <char position expression> is specified, then

Case:

a) If the first <string value expression> has a length of 0, then the result is 1.

b) If the value of the first <string value expression> is equal to an identical-length substring of contiguous characters or bits from the value of the second <string value expression>, then the result is 1 greater than the number of characters or bits within the value of the second <string value expression> preceding the start of the first such substring.

c) Otherwise, the result is 0.

3) If <blob position expression> is specified, then:

Case:

a) If the first <blob value expression> has a length of 0, then the result is 1.

b) If the value of the first <blob value expression> is equal to an identical-length substring of contiguous octets from the value of the second <blob value expression>, then the result is 1 greater than the number of octets within the value of the second <blob value expression> preceding the start of the first such substring.

c) Otherwise, the result is 0.

4) If <extract expression> is specified, then

Case:

a) If <extract field> is a <datetime field>, then the result is the value of the datetime field identified by that <datetime field> and has the same sign as the <extract source>.

Note: If the value of the identified <datetime field> is zero or if <extract source> is not an <interval value expression>, then the sign is irrelevant.

b) Otherwise, let *TZ* be the interval value of the implicit or explicit time zone associated with the <datetime value expression>. If <extract field> is TIMEZONE_HOUR, then the result is calculated as

EXTRACT (HOUR FROM *TZ*)

Otherwise, the result is calculated as

EXTRACT (MINUTE FROM *TZ*)

5) If a <char length expression> is specified, then let *S* be the <string value expression>.

Case:

a) If the data type of *S* is character string, then the result is the number of characters in the value of *S*.

b) Otherwise, the result is OCTET_LENGTH(*S*).

6) If an <octet length expression> is specified, then let *S* be the <string value expression>. The result of the <octet length expression> is the smallest integer not less than the quotient of the division (BIT_LENGTH(*S*)/8).

7) If a <bit length expression> is specified, then let *S* be the <string value expression>. The result of the <bit length expression> is the number of bits in the value of *S*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <position expression> shall not be a <blob position expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <numeric value function> shall not be a <position expression>.

   b) A <numeric value function> shall not contain a <length expression> that is a <bit length expression>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <numeric value function> shall not be a <length expression>.

   b) A <numeric value function> shall not be an <extract expression>.

6.10 <string value function>

Function

Specify a function yielding a value of type character string or bit string.

Format

```
<string value function> ::=
        <character value function>
      | <bit value function>


      | <blob value function> <character value function> ::=
        <character substring function>
      | <regular expression substring function>
      | <fold>
      | <form-of-use conversion>
      | <character translation>
      | <trim function>
      | <character overlay function>

<character substring function> ::=
      SUBSTRING <left paren> <character value expression> FROM <start position>
                  [ FOR <string length> ] <right paren>

<regular expression substring function> ::=
      SUBSTRING <left paren> <character value expression> FROM
                  <character value expression> FOR <escape character> <right paren>

<fold> ::= { UPPER | LOWER } <left paren> <character value expression> <right paren>

<form-of-use conversion> ::=
      CONVERT <left paren> <character value expression>
          USING <form-of-use conversion name> <right paren>

<character translation> ::=
      TRANSLATE <left paren> <character value expression>
          USING <translation name> <right paren>

<trim function> ::=
      TRIM <left paren> <trim operands> <right paren>

<trim operands> ::=
      [ [ <trim specification> ] [ <trim character> ] FROM ] <trim source>

<trim source> ::= <character value expression>

<trim specification> ::=
        LEADING
      | TRAILING
      | BOTH

<trim character> ::= <character value expression>

<character overlay function> ::=
      OVERLAY <left paren> <character value expression>
        PLACING <character value expression>
        FROM <start position>
```

```
        [ FOR <string length> ] <right paren>

<blob value function> ::=
        <blob substring function>
      | <blob trim function>
      | <blob overlay function>

<blob substring function> ::=
      SUBSTRING <left paren> <blob value expression> FROM <start position>
        [ FOR <string length> ] <right paren>

<blob trim function> ::=
      TRIM <left paren> <blob trim operands> <right paren>

<blob trim operands> ::=
      [ [ <trim specification> ] [ <trim octet> ] FROM ] <blob trim source>

<blob trim source> ::= <blob value expression>

<trim octet> ::= <blob value expression>

<blob overlay function> ::=
      OVERLAY <left paren> <blob value expression>
        PLACING <blob value expression>
        FROM <start position>
        [ FOR <string length> ] <right paren>

<bit value function> ::=
      <bit substring function>

<bit substring function> ::=
      SUBSTRING <left paren> <bit value expression> FROM <start position>
          [ FOR <string length> ] <right paren>

<start position> ::= <numeric value expression>

<string length> ::= <numeric value expression>
```

Syntax Rules

1) The data type of a <start position> and <string length> shall be exact numeric with scale 0.

2) If <character substring function> or <character overlay function> is specified, then:

   a) The data type of the <character substring function> is variable-length character string with maximum length equal to the fixed length or maximum variable length of the <character value expression>. The character repertoire and form-of-use of the <character substring function> are the same as the character repertoire and form-of-use of the <character value expression>.

   b) The collating sequence and the coercibility attribute are determined as specified for monadic operators in Subclause 4.2.3, "Rules determining collating sequence usage", where the first operand of SUBSTRING plays the role of the monadic operand.

3) If <regular expression substring function> is specified, then the data types of the <escape character> and the <character value expression>s of the <regular expression substring function> shall be character string with the same character repertoire; the data type of the <regular expression substring function> shall be character string with the same character repertoire. The value of the <escape character> shall have length 1.

4) If <fold> is specified, then:

   a) The data type of the result of <fold> is the data type of the <character value expression>.

   b) The collating sequence and the coercibility attribute are determined as specified for monadic operators in Subclause 4.2.3, "Rules determining collating sequence usage", where the operand of the <fold> is the monadic operand.

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

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

   b) The data type of the result is variable-length character string with implementation-defined maximum length. The character set of the result is the same as the character repertoire of the <character value expression> and form-of-use determined by the form-of-use conversion identified by the <form-of-use conversion name>. Let *CR* be that character repertoire. The result has the *Implicit* coercibility attribute and its collating sequence is *X*, where *X* is the default collating sequence of *CR*.

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

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

   b) The data type of the <character translation> is variable-length character string with implementation-defined maximum length and character repertoire equal to the character repertoire of the target character set of the translation. Let *CR* be that character repertoire. The result has the *Implicit* coercibility attribute and its collating sequence is *X*, where *X* is the default collating sequence of *CR*.

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

   a) If FROM is specified, then either <trim specification> or <trim character> or both shall be specified.

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

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

   d) If

      TRIM (*SRC*)

      is specified, then

      TRIM (BOTH ' ' FROM *SRC*)

      is implicit.

   e) The data type of the <trim function> is variable-length character string with maximum length equal to the fixed length or maximum variable length of the <trim source>.

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

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

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

8) If <blob substring function> or <blob overlay function> is specified, then the data type of the <blob substring function> is binary string with maximum length equal to the maximum length of the <blob value expression>.

9) If <blob trim function> is specified, then:

a) If FROM is specified, then either <trim specification> or <trim octet> or both shall be specified.

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

c) If <trim character> is not specified, then BIN'00' is implicit.

d) If

TRIM (SRC)

is specified, then

TRIM (BOTH BIN'00' FROM SRC)

is implicit.

e) The data type of the <blob trim function> is binary string with maximum length equal to the maximum length of the <blob trim source>.

10) If <bit substring function> is specified, then the data type of the <bit substring function> is variable-length bit string with maximum length equal to the fixed length or maximum variable length of the <bit value expression>.

11) The null class of a <character value function> is determined as follows:

Case:

a) In a <character substring function> or in a <character overlay function>,

Case:

i) If those of <character value expression>, <start position> and <string length> that do not have the general null class have the same null class, then the result has that null class.

ii) Otherwise, the result has the general null class.

b) In a <regular expression substring function>,

Case:

i) If those out of <character value expression>s and <escape character> that do not have the general null class have the same null class, then the result has that null class.

ii) Otherwise, the result has the general null class.

12) The null class of a <blob substring function> or a <blob overlay function> is determined as follows:

Case:

a) If those of <blob value expression>, <start position> and <string length> that do not have the general null class have the same null class, then the result has that null class.

b) Otherwise, the result has the general null class.

Access Rules

1) The applicable privileges shall include USAGE for every <translation name> contained in the <string value expression>.

General Rules

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

a) Let *C* be the value of the <character value expression>, let *LC* be the length of *C*, and let *S* be the value of the <start position>.

b) If <string length> is specified, then let *L* be the value of <string length> and let *E* be *S*+*L*. Otherwise, let *E* be the larger of *LC* + 1 and *S*.

c) If either *C*, *S*, or *L* is a null value, then the result of the <character substring function> is the general null value.

d) If *E* is less than *S*, then an exception condition is raised: *data exception—substring error*.

e) Case:

i) If *S* is greater than *LC* or if *E* is less than 1, then the result of the <character substring function> is a zero-length string.

ii) Otherwise,

1) Let *S1* be the larger of *S* and 1. Let *E1* be the smaller of *E* and *LC*+1. Let *L1* be *E1*−*S1*.

2) The result of the <character substring function> is a character string containing the *L1* characters of *C* starting at character number *S1* in the same order that the characters appear in *C*.

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

i) Let *C* be the result of the first <character value expression>, let *R* be the result of the second <character value expression>, and let *E* be the result of the <escape character>.

ii) If one or more of *C*, *R* or *E* is a null value, then,

Case:

1) If the result has the general null class, then the result of the <regular expression substring function> is the general null value.

2) Otherwise the result of the <regular expression substring function> is the minimum of the null values of the operands.

iii) *R* is divided into three substring parts, *R1*, *R2*, and *R3*, such that *R1*, *R2*, and *R3* are regular expressions and

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

is true. If an <escape character> is specified, then

'*C*' SIMILAR TO '*R1*' || '*R2*' || '*R3*'

is true.

iv) If

'*C*' SIMILAR TO '*R1*' || '*R2*' || '*R3*' ESCAPE '*E*'

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

v) If *C* is not a null value, then the result *S* of the <regular expression substring function> satisfies the following conditions:

1) *S* is a substring of *C* such that there are substrings *S1* and *S2* and

'*C*' = '*S1*' || '*S*' || '*S2*'

is true.

2) '*S1*' SIMILAR TO '*R1*' ESCAPE '*E*'

is true and for any substring *S* of *S1*,

'*S*' SIMILAR TO '*R1*' ESCAPE '*E*'

is false.

3) '*S*' SIMILAR TO '*R2*' ESCAPE '*E*'

is true and for any substring *S* of *S*,

'*S*' SIMILAR TO '*R2*' ESCAPE '*E*'

is false.

3) If <fold> is specified, then:

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

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

c) Case:

i) If UPPER is specified, then the result of the <fold> is a copy of *S* in which every <simple Latin lower case letter> that has a corresponding <simple Latin upper case letter> in the character repertoire of *S* is replaced by that <simple Latin upper case letter>.

ii) If LOWER is specified, then the result of the <fold> is a copy of *S* in which every <simple Latin upper case letter> that has a corresponding <simple Latin lower case letter> in the character repertoire of *S* is replaced by that <simple Latin lower case letter>.

4) If a <character translation> is specified, then

Case:

a) If the value of <character value expression> is a null value, then the result of the <character translation> is the general null value.

b) Otherwise, the value of the <character translation> is the value of the <character value expression> after translation to the character repertoire of the target character set of the translation.

5) If a <form-of-use conversion> is specified, then

Case:

a) If the value of <character value expression> is a null value, then the result of the <form-of-use conversion> is the general null value.

b) Otherwise, the value of the <form-of-use conversion> is the value of the <character value expression> after the application of the form-of-use conversion specified by <form-of-use conversion name>.

6) If <trim function> is specified, then:

a) Let S be the value of the <trim source>.

b) If <trim character> is specified, then let SC be the value of <trim character>; otherwise, let SC be <space>.

c) If either S or SC is a null value, then the result of the <trim function> is the general null value.

d) If the length in characters of SC is not 1, then an exception condition is raised: *data exception—trim error*.

e) Case:

i) If BOTH is specified or if no <trim specification> is specified, then the result of the <trim function> is the value of S with any leading or trailing characters equal to SC removed.

ii) If TRAILING is specified, then the result of the <trim function> is the value of S with any trailing characters equal to SC removed.

iii) If LEADING is specified, then the result of the <trim function> is the value of S with any leading characters equal to SC removed.

7) If <blob substring function> is specified, then

a) Let B be the value of the <blob value expression>, let LB be the length in octets of B, and let S be the value of the <start position>.

b) If <string length> is specified, then let L be the value of <string length> and let E be $S + L$. Otherwise, let E be the larger of $LB + 1$ and S.

c) If either B, S, or L is a null value, then the result of the <blob substring function> is the general null value.

d) If E is less than S, then an exception condition is raised: *data exception—substring error*.

e) Case:

i) If S is greater than LB or if E is less than 1, then the result of the <blob subtring function> is a zero-length string.

ii) Otherwise:

1) Let $S1$ be the larger of S and 1. Let $E1$ be the smaller of E and $LB + 1$. Let $L1$ be $E1 - S1$.

2) The result of the <blob substring function> is a binary large object string containing $L1$ octets of B starting at octe number $S1$ in the same order that the octets appear in B.

8) If <blob trim function> is specified, then

a) Let S be the value of the <trim source>.

b) If <trim octet> is specified, then let SO be the value of <trim octet>; otherwise let SO be X'00'.

c) If either S or SO a null value, then the result of the <blob trim function> is the general null value.

d) If the length in octets of SO is not 1, then an exception condition is raised: *data exception—trim error*.

e) Case:

i) If BOTH is specified or if no <trim specification> is specified, then the result of the <blob trim function> is the value of S with any leading or trailing octets equal to SO removed.

ii) If TRAILING is specified, then the result of the <blob trim function> is the value of S with any trailing octets equal to SO removed.

iii) If LEADING is specified, then the result of the <blob trim function> is the value of S with any leading octets equal to SO removed.

9) If <bit substring function> is specified, then:

a) Let B be the value of the <bit value expression>, let LB be the length in bits of B, and let S be the value of the <start position>.

b) If <string length> is specified, then let L be the value of <string length> and let E be $S+L$. Otherwise, let E be the larger of $LB + 1$ and S.

c) If either B, S, or L is a null value, then the result of the <bit substring function> is the general null value.

d) If E is less than S, then an exception condition is raised: *data exception—substring error*.

e) Case:

i) If S is greater than LB or if E is less than 1, then the result of the <bit substring function> is a zero-length string.

ii) Otherwise,

1) Let *S1* be the larger of *S* and 1. Let *E1* be the smaller of *E* and *LB*+1. Let *L1* be *E1*−*S1*.

2) The result of the <bit substring function> is a bit string containing *L1* bits of *B* starting at bit number *S1* in the same order that the bits appear in *B*.

10) If <character overlay function> is specified, then let *CV* be the <character value expression>, let *SP* be the <start position>, let *SL* be the <string length>, and let *RS* be the <character value expression> following PLACING. The <character overlay function> is equivalent to

SUBSTRING (*CV* FROM 1 FOR *SP* − 1)
|| *RS*
|| SUBSTRING (*CV* FROM *SP* + *SL*)

11) If <blob overlay function> is specified, then let *BV* be the <blob value expression>, let *SP* be the <start position>, let *SL* be the <string length>, and let *RS* be the <blob value expression> following PLACING. The <blob overlay function> is equivalent to

SUBSTRING (*BV* FROM 1 FOR *SP* − 1)
|| *RS*
|| SUBSTRING (*BV* FROM *SP* + *SL*)

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <string value function> shall not be a <regular expression substring function>, a <blob value function>, or a <character overlay function>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <character value function> shall not be a <fold>.

b) Conforming Intermediate SQL language shall contain no <character translation>.

c) Conforming Intermediate SQL language shall contain no <form-of-use conversion>.

d) Conforming Intermediate SQL language shall contain no <bit value function>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <character value function> shall not be a <character substring function>.

b) A <character value function> shall not be a <trim function>.

6.11 <datetime value function>

Function

Specify a function yielding a value of type datetime.

Format

```
<datetime value function> ::=
        <current date value function>
      | <current time value function>
      | <current timestamp value function>

<current date value function> ::= CURRENT_DATE

<current time value function> ::=
        CURRENT_TIME [ <left paren> <time precision> <right paren> ]

<current timestamp value function> ::=
        CURRENT_TIMESTAMP [ <left paren> <timestamp precision> <right paren> ]
```

Syntax Rules

1) The data type of a <current date value function> is DATE. The data type of a <current time value function> is TIME WITH TIME ZONE. The data type of a <current timestamp value function> is TIMESTAMP WITH TIME ZONE.

   Note: See the Syntax Rules of Subclause 6.1, "<data type>", for rules governing <time precision> and <timestamp precision>.

2) The null class of a <datetime value function> is the general null class.

Access Rules

None.

General Rules

1) The <datetime value function>s CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP respectively return the current date, current time, and current timestamp; the time and timestamp values are returned with time zone displacement equal to the current time zone displacement of the SQL-session.

2) If specified, <time precision> and <timestamp precision> respectively determine the precision of the time or timestamp value returned.

3) If an SQL-statement causes the evaluation of one or more <datetime value function>s, then all such evaluations are effectively performed simultaneously. The time of evaluation of the <datetime value function> during the execution of the SQL-statement is implementation-dependent.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <time precision> or <timestamp precision>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <datetime value function>.

6.12 <OID value function>

Function

Specify a function yielding the object identifier of an instance of an object abstract data type.

Format

```
<OID value function> ::=
      OID { <item reference> | <attribute reference> }
```

Syntax Rules

1) The data type of the <item reference> or <attribute reference> shall be an object abstract data type *ADTN* whose elaboration mode is INSTANCE.

2) The data type of the result of the <OID value function> is *ADTN* with no <elaboration mode>.

Access Rules

None.

General Rules

1) Let *V* be the column, parameter, variable, or attribute referenced by the <item reference> or <attribute reference>.

2) The result of the <OID value function> is the object identifier associated with *V*

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <OID value function>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.13 <case expression>

Function

Specify a conditional value.

Format

```
<case expression> ::=
        <case abbreviation>
      | <case specification>

<case abbreviation> ::=
        NULLIF <left paren> <value expression> <comma>
              <value expression> <right paren>
      | COALESCE <left paren> <value expression>
              { <comma> <value expression> }... <right paren>

<case specification> ::=
        <simple case>
      | <searched case>

<simple case> ::=
      CASE <case operand>
        <simple when clause>...
        [ <else clause> ]
      END

<searched case> ::=
      CASE
        <searched when clause>...
        [ <else clause> ]
      END

<simple when clause> ::= WHEN <when operand> THEN <result>

<searched when clause> ::= WHEN <search condition> THEN <result>

<else clause> ::= ELSE <result>

<case operand> ::= <value expression>

<when operand> ::= <value expression>

<result> ::= <result expression> | NULL

<result expression> ::= <value expression>
```

Syntax Rules

1) NULLIF (V_1, V_2) is equivalent to the following <case specification>:

   CASE WHEN V_1=V_2 THEN NULL ELSE V_1 END

2) COALESCE (V_1, V_2) is equivalent to the following <case specification>:

   CASE WHEN V_1 IS NOT NULL THEN V_1 ELSE V_2 END

3) COALESCE (V_1, V_2, . . . ,V_n), for n >= 3, is equivalent to the following <case specification>:

   CASE WHEN V_1 IS NOT NULL THEN V_1 ELSE COALESCE (V_2, . . . ,V_n) END

4) If a <case specification> specifies a <simple case>, then let *CO* be the <case operand>:

   a) The data type of each <when operand> *WO* shall be comparable with the data type of the <case operand>.

   b) The <case specification> is equivalent to a <searched case> in which each <searched when clause> specifies a <search condition> of the form "*CO*=*WO*".

5) At least one <result> in a <case specification> shall specify a <result expression>.

6) If an <else clause> is not specified, then ELSE NULL is implicit.

7) The data type of a <case specification> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of all <result expression>s in the <case specification>.

8) The null class of a <case expression> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the null classes of all result expressions in the <case expression>.

Access Rules

None.

General Rules

1) Case:

   a) If a <result> specifies NULL, then its value is the general null value.

   b) If a <result> specifies a <value expression>, then its value is the value of that <value expression>.

2) Case:

   a) If the <search condition> of some <searched when clause> in a <case specification> is true, then the value of the <case specification> is the value of the <result> of the first (leftmost) <searched when clause> whose <search condition> is true, cast as the data type of the <case specification>.

   b) If no <search condition> in a <case specification> is true, then the value of the <case expression> is the value of the <result> of the explicit or implicit <else clause>, cast as the data type of the <case specification>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <case expression>.

6.14 <cast specification>

Function

Specify a data conversion.

Format

```
<cast specification> ::=
       CAST <left paren> <cast operand> AS <cast target> <right paren>

<cast operand> ::=
         <value expression>
       | NULL

<cast target> ::=
         <domain name>
       | <data type>
```

Syntax Rules

1) Case:

   a) If a <domain name> is specified, then let *TD* be the <data type> of the specified domain.

   b) If a <data type> is specified, then let *TD* be the specified <data type>.

2) The data type of the result of the <cast specification> is *TD*.

3) If the <cast operand> is a <value expression>, then let *SD* be the underlying data type of the <value expression>.

4) If the <cast operand> is a <value expression> and neither *TD* nor *SD* is a collection type, then the valid combinations of *TD* and *SD* in a <cast specification> are given by the following table. "Y" indicates that the combination is syntactically valid without restriction; "M" indicates that the combination is valid subject to other Syntax Rules in this Subclause being satisfied; and "N" indicates that the combination is not valid:

<data type> SD of <value expression>	<data type> of TD																
	EN	AN	VC	FC	VB	FB	D	T	TS	YM	DT	ET	BO	ADT	NT	CL	BL
EN	Y	Y	Y	Y	N	N	N	N	N	M	M	Y	N	M	M	Y	N
AN	Y	Y	Y	Y	N	N	N	N	N	N	N	N	N	M	M	Y	N
C	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	M	M	Y	N
B	N	N	Y	Y	Y	Y	N	N	N	N	N	N	N	M	M	Y	N
D	N	N	Y	Y	N	N	Y	N	Y	N	N	N	N	M	M	Y	N
T	N	N	Y	Y	N	N	N	Y	Y	N	N	N	N	M	M	Y	N
TS	N	N	Y	Y	N	N	Y	Y	Y	N	N	N	N	M	M	Y	N
YM	M	N	Y	Y	N	N	N	N	N	Y	N	N	N	M	M	Y	N
DT	M	N	Y	Y	N	N	N	N	N	N	Y	N	N	M	M	Y	N
ET	Y	N	Y	Y	N	N	N	N	N	N	N	Y	N	M	M	Y	N
BO	N	N	Y	Y	N	N	N	N	N	N	N	N	Y	M	M	Y	N
ADT	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M	M
NT	M	M	M	M	M	M	M	M	M	M	M	M	M	M	N	M	N

```
    BL        N   N   N   N   N   N   N   N   N   N   N   N   N   M   N   N   Y
```

Where:

```
    EN = Exact Numeric
    AN = Approximate Numeric
    C  = Character (Fixed- or Variable-length, or character large object)
    FC = Fixed-length Character
    VC = Variable-length Character
    CL = Character Large Object
    B  = Bit String (Fixed- or Variable-length)
    FB = Fixed-length Bit String
    VB = Variable-length Bit String
    D  = Date
    T  = Time
    TS = Timestamp
    YM = Year-Month Interval
    DT = Day-Time Interval
    ET = Enumerated Type
    BO = Boolean
   ADT = Abstract Data Type
    NT = any distinct type
    BL = Binary Large Object
```

5) If *TD* is an interval and *SD* is exact numeric, then *TD* shall contain only a single <datetime field>.

6) If *TD* is exact numeric and *SD* is an interval, then *SD* shall contain only a single <datetime field>.

7) If *SD* is character string and *TD* is fixed-length, variable-length, or large object character string, then the character repertoires of *SD* and *TD* shall be the same.

8) If *TD* is a fixed-length, variable-length or large object character string, then the collating sequence of the result of the <cast specification> is the default collating sequence for the character repertoire of *TD* and the result of the <cast specification> has the *Coercible* coercibility attribute.

9) If either *SD* or *TD* is an abstract data type, then:

   a) There shall be an abstract data type descriptor that identifies a cast function *CP* that has an operand data type *SD* and a return data type *TD*; or

   b) One of *SD* or *TD* shall be a character string type and the other shall be an abstract data type described by an abstract data type descriptor that indicates that the abstract data type is WITH OID VISIBLE.

10) If either *SD* or *TD* is a distinct type, then exactly one of *SD* or *TD* shall be a distinct type and the other shall be the source type of that distinct type.

    Note: *Source type* is defined in Subclause 11.49, "<distinct type definition>".

11) If *TD* is a collection type, then *SD* shall be a collection type.

12) If *SD* is a collection type, then let *ESD* be the <data type> of the elements of type *SD*.

13) If *TD* is a collection type, then let *ETD* be the <data type> of the elements of type *TD*.

14) If both *SD* and *TD* are collection types, then a CAST of a <value expression> of <data type> *ESD* to <data type> *ETD* shall be permitted by the Syntax Rules of this Subclause.

15) If *SD* is a collection type, then either *TD* shall be the same data type as *ESD*, or both *SD* and *TD* shall be the same collection type, or *SD* shall be a multiset type and *TD* shall be a set type or a list type.

16) If <domain name> is specified, then let *D* be the domain identified by the <domain name>. If the <cast specification> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <domain name> shall include the descriptor of *D*. If the <cast specification> is contained in a <schema defintion> *S*, then the schema identified by the explicit or implicit qualifier of the <domain name> shall include the descriptor of *D* or *S* shall include a <schema element> that creates the descriptor of *D*.

ANSI Only–SQL3

17) If either *SD* or *TD* is a row identifier type, then either:

a) They shall be the same row identifier type, or

b) *SD* shall be a subtype of *TD*.

Access Rules

1) If <domain name> is specified, then the applicable privileges shall include USAGE.

General Rules

1) If the <cast operand> is a <value expression>, then let *SV* be its value.

2) Case:

a) If the <cast operand> specifies NULL, then *TV* is the general null value.

b) If *SV* is a null value, then

Case:

i) If a <domain name> is specified that has the same null class as the <cast operand>, then the result is that null value.

ii) Otherwise the result is the general null value.

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

3) If either *SD* or *TD* is a distinct type, then

Case:

a) If *SD* is a distinct type, then *TV* is the representation of *SV* in the data type of *TD*.

b) Otherwise, let *BD* be the source type of the distinct type *TD*. *TV* is the representation of the result of:

CAST (*SV* AS *BD*)

in the data type of *TD*.

4) If either *TD* or *SD* is an abstract data type, then

Case:

a) If a cast function *CP* that casts *SD* to *TD* exists, then invoke *CP* with a <routine invocation>, as appropriate, with <argument> *SV* to obtain a result *TR*.

Note: <routine invocation> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

Case:

i) If *TR* is a null value, then the result of the <cast specification> is the null value.

ii) If *TR* is not a null value, then let *TV* be the non-null value of *TR*.

b) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid instance of the data type *TD*, then let *TV* be the object identifier of that instance.

ii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

c) If *TD* is fixed-length character string, then let *LTD* be the length in characters of *TD* and let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", such that the interpreted value of *Y* is *SV*.

Case:

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

ii) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

iii) If the length in characters *LY* of *Y* is less than *LTD*, then TV is *Y* extended on the right by *LTD* – *LY* <space>s.

iv) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

d) If *TD* is variable-length character string or large object character string, then let *MLTD* be the maximum length in characters of *TD* and let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", such that the interpreted value of *Y* is *SV*.

Case:

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

ii) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

5) If *SD* is a collection data type, then

Case:

a) If *TD* is the same data type as *ESD*, then

Case:

i) If *SV* is an empty collection, then *TV* is the general null value.

ii) If the cardinality of *SV* is exactly one, then *TV* is the one element of *SV*.

iii) If the cardinality of *SV* is greater than one, then an exception condition is raised: *cardinality violation*.

b) If *SD* is a multiset type and *TD* is a multiset type, then *TV* is a multiset with the same number of elements as *SV*. Each element of *TV* is created by casting the corresponding element of *SV* to the data type *ETD*.

c) If *SD* is a set type and *TD* is a set type, then *TV* is a set with the same number of elements as *SV*. Each element of *TV* is created by casting the corresponding element of *SV* to the data type *ETD*.

d) If *SD* is a list type and *TD* is a list type, then *TV* is a list with same number of elements as *SV*. Each element of *TV* is created by casting the corresponding element of *SV* to the data type *ETD*. The order of elements in *TV* is the same as the order of their corresponding elements in *SV*.

e) If *SD* is a multiset type and *TD* is a set type, then *TV* is a set where the number of elements is equal to the number of elements that would remain in *SV* if redundant duplicate elements were removed. Each element of *TV* is created by casting the corresponding element (after redundant duplicates are removed) of *SV* to the data type *ETD*.

f) If *SD* is a multiset type and *TD* is a list type, then *TV* is a list with the same number of elements as *SV*. Each element of *TV* is created by casting the corresponding element of *SV* to the data type *ETD*. The order of elements in *TV* is implementation-dependent.

6) If *TD* is exact numeric, then

Case:

a) If *SD* is exact numeric or approximate numeric, then

Case:

i) If there is a representation of *SV* in the data type *TD* that does not lose any leading significant digits after rounding or truncating if necessary, then *TV* is that representation. The choice of whether to round or truncate is implementation-defined.

ii) Otherwise, an exception condition is raised: *data exception—numeric value out of range*.

b) If *SD* is character string, then *SV* is replaced by *SV* with any leading or trailing <space>s removed.

Case:

i) If *SV* does not comprise a <signed numeric literal> as defined by the rules for <literal> in Subclause 5.3, "<literal>", then an exception condition is raised: *data exception—invalid character value for cast*.

ii) Otherwise, let *LT* be that <signed numeric literal>. The <cast specification> is equivalent to

CAST (*LT* AS *TD*)

c) If *SD* is an interval data type, then

Case:

i) If there is a representation of *SV* in the data type *TD* that does not lose any leading significant digits, then *TV* is that representation.

ii) Otherwise, an exception condition is raised: *data exception—numeric value out of range*.

d) If *SD* is an enumerated type, then *TV* is the position number of *SV* in *SD*.

7) If *TD* is approximate numeric, then

Case:

a) If *SD* is exact numeric or approximate numeric, then

Case:

i) If there is a representation of *SV* in the data type *TD* that does not lose any leading significant digits after rounding or truncating if necessary, then *TV* is that representation. The choice of whether to round or truncate is implementation-defined.

ii) Otherwise, an exception condition is raised: *data exception—numeric value out of range*.

b) If *SD* is character string, then *SV* is replaced by *SV* with any leading or trailing <space>s removed.

Case:

i) If *SV* does not comprise a <signed numeric literal> as defined by the rules for <literal> in Subclause 5.3, "<literal>", then an exception condition is raised: *data exception—invalid character value for cast*.

ii) Otherwise, let *LT* be that <signed numeric literal>. The <cast specification> is equivalent to

CAST (*LT* AS *TD*)

8) If *TD* is fixed-length character string, then let *LTD* be the length in characters of *TD*.

Case:

a) If *SD* is exact numeric, then let *YP* be the shortest character string that conforms to the definition of <exact numeric literal> in Subclause 5.3, "<literal>", whose scale is the same as the scale of *SD* and whose interpreted value is the absolute value of *SV*.

If *SV* is less than 0, then let *Y* be the result of

'–' || *YP*

Otherwise, let *Y* be *YP*.

Case:

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

ii) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

iii) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD*–*LY* <space>s.

iv) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

b) If *SD* is approximate numeric, then:

i) Let *YP* be a character string as follows:

Case:

1) If *SV* equals 0, then *YP* is '0E0'.

2) Otherwise, *YP* is the shortest character string that conforms to the definition of <approximate numeric literal> in Subclause 5.3, "<literal>", whose interpreted value is equal to the absolute value of *SV* and whose <mantissa> consists of a single <digit> that is not '0', followed by a <period> and an <unsigned integer>.

ii) If *SV* is less than 0, then let *Y* be the result of

'–' || *YP*

Otherwise, let *Y* be *YP*.

iii) Case:

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

2) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

3) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD*–*LY* <space>s.

4) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

c) If *SD* is fixed-length character string, variable-length character string, or large object character string, then

Case:

i) If the length in characters of *SV* is equal to *LTD*, then *TV* is *SV*.

ii) If the length in characters of *SV* is larger than *LTD*, then *TV* is the first *LTD* characters of *SV*. If any of the remaining characters of *SV* are non-<space> characters, then a completion condition is raised: *warning—string data, right truncation*.

iii) If the length in characters *M* of *SV* is smaller than *LTD*, then *TV* is *SV* extended on the right by *LTD*−M <space>s.

d) If *SD* is a fixed-length bit string or variable-length bit string, then let *LSV* be the value of BIT_LENGTH(SV) and let *B* be the BIT_LENGTH of the character with the smallest BIT_LENGTH in the form-of-use of *TD*. Let *PAD* be the value of the remainder of the division *LSV*/*B*. Let *NC* be a character whose bits all have the value 0.

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

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

TV is the result of

SUBSTRING (*SVC* ‖ *LTDS* FROM 1 FOR *LTD*)

Case:

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

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

e) If *SD* is a datetime data type or an interval data type, then let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", and such that the interpreted value of *Y* is *SV* and the interpreted precision of *Y* is the precision of *SD*.

Case:

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

ii) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

iii) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD*−*LY* <space>s.

iv) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

f) If *SD* is an enumerated type, then let *Y* be a character string that is the same as the <enumeration name> *SV* in *SD*.

Case:

i) If the length in characters *LY* of *Y* is equal to *LTD*, then *TV* is *Y*.

ii) If the length in characters *LY* of *Y* is less than *LTD*, then *TV* is *Y* extended on the right by *LTD*−*LY* <space>s.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

g) If *SD* is boolean, then

Case:

i) If *SV* is *true* and *LTD* is not less than 4, then *TV* is 'TRUE' extended on the right by *LTD*−4 <space>s.

ii) If *SV* is *false* and *LTD* is not less than 5, then *TV* is 'FALSE' extended on the right by *LTD*−5 <space>s.

iii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

9) If *TD* is variable-length character string or large object character string, then let *MLTD* be the maximum length in characters of *TD*.

Case:

a) If *SD* is exact numeric, then let *YP* be the shortest character string that conforms to the definition of <exact numeric literal> in Subclause 5.3, "<literal>", whose scale is the same as the scale of *SD* and whose interpreted value is the absolute value of *SV*.

If *SV* is less than 0, then let *Y* be the result of

'−' || *YP*

Otherwise, let *Y* be *YP*.

Case:

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

ii) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

b) If *SD* is approximate numeric, then

i) Let *YP* be a character string as follows:

Case:

1) If *SV* equals 0, then *YP* is '0E0'.

2) Otherwise, *YP* is the shortest character string that conforms to the definition of <approximate numeric literal> in Subclause 5.3, "<literal>", whose interpreted value is equal to the absolute value of *SV* and whose <mantissa> consists of a single <digit> that is not '0', followed by a <period> and an <unsigned integer>.

ii) If *SV* is less than 0, then let *Y* be the result of

'−' || *YP*

Otherwise, let *Y* be *YP*.

iii) Case:

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

2) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

3) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

c) If *SD* is fixed-length character string, variable-length character string, or large object character string, then

Case:

i) If the length in characters of *SV* is less than or equal to *MLTD*, then *TV* is *SV*.

ii) If the length in characters of *SV* is larger than *MLTD*, then *TV* is the first *MLTD* characters of *SV*. If any of the remaining characters of *SV* are non-<space> characters, then a completion condition is raised: *warning—string data, right truncation*.

d) If *SD* is a fixed-length bit string or variable-length bit string, then let *LSV* be the value of BIT_LENGTH(SV) and let *B* be the BIT_LENGTH of the character with the smallest BIT_LENGTH in the form-of-use of *TD*. Let *PAD* be the value of the remainder of the division *LSV*/*B*.

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

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

Case:

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

ii) Otherwise, *TV* is the result of

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

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

e) If *SD* is a datetime data type or an interval data type then let *Y* be the shortest character string that conforms to the definition of <literal> in Subclause 5.3, "<literal>", and such that the interpreted value of *Y* is *SV* and the interpreted precision of *Y* is the precision of *SD*.

Case:

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

ii) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

f) If *SD* is an enumerated type, then let *Y* be a character string that is the same as the <enumeration name> *SV* in *D*.

Case:

i) If the length in characters *LY* of *Y* is less than or equal to *MLTD*, then *TV* is *Y*.

ii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

g) If *SD* is boolean, then

Case:

i) If *SV* is *true* and *MLTD* is not less than 4, then *TV* is 'TRUE'.

ii) If *SV* is *false* and *MLTD* is not less than 5, then *TV* is 'FALSE'.

iii) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

10) If *TD* and *SD* are binary string data types, then let *MLTD* be the maximum length in octets of *TD*.

Case:

a) If the length in octets of *SV* is less than or equal to *MLTD*, then *TV* is *SV*.

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

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

Case:

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

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

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

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

Case:

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

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

13) If *TD* is the datetime data type DATE, then

Case:

a) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> or for <unquoted date string> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

ii) If a <datetime value> does not conform to the natural rules for dates or times according to the Gregorian calendar, then an exception condition is raised: *data exception—invalid datetime format*.

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

b) If *SD* is a date, then *TV* is *SV*.

c) If *SD* is a timestamp, then *TV* is the year, month, and day <datetime field>s of *SV* adjusted to the implicit or explicit time zone displacement of *SV*.

14) If *TD* is the datetime data type TIME, then

Case:

a) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) 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 *TD*, then let *TV* be that value.

ii) If a <datetime value> does not conform to the natural rules for dates or times according to the Gregorian calendar, then an exception condition is raised: *data exception—invalid datetime format*.

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

b) If *SD* is a time, then *TV* is *SV*. If *TD* is specified WITH TIME ZONE, then *TV* also includes the implicit or explicit time zone displacement of *SV*; otherwise, *TV* is adjusted to the current time zone displacement of the SQL-session.

c) If *SD* is a timestamp, then *TV* is the hour, minute, and second <datetime field>s of *SV*. If *TD* is specified WITH TIME ZONE, then *TV* also includes the implicit or explicit time zone displacement of *SV*; otherwise, *TV* is adjusted to the current time zone displacement of the SQL-session.

15) If *TD* is the datetime data type TIMESTAMP, then

Case:

a) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> or for <unquoted timestamp string> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

ii) If a <datetime value> does not conform to the natural rules for dates or times according to the Gregorian calendar, then an exception condition is raised: *data exception—invalid datetime format*.

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

b) If *SD* is a date, then the <datetime field>s hour, minute, and second of *TV* are set to 0 and the <datetime field>s year, month, and day of *TV* are set to their respective values in *SV*. If *TD* is specified WITH TIME ZONE, then the time zone fields of *TV* are set to the current time zone displacement of the SQL-session.

c) If *SD* is a time, then the <datetime field>s year, month, and day of *TV* are set to their respective values in an execution of CURRENT_DATE and the <datetime field>s hour, minute, and second of *TV* are set to their respective values in *SV*. If *TD* is specified WITH TIME ZONE, then the time zone fields of *TV* are set to the explicit or implicit time zone interval of *SV*.

d) If *SD* is a timestamp, then *TV* is *SV*.

16) If *TD* is interval, then

Case:

a) If *SD* is exact numeric, then

Case:

i) If the representation of *SV* in the data type *TD* would result in the loss of leading significant digits, then an exception condition is raised: *data exception—interval field overflow*.

ii) Otherwise, *TV* is that representation.

b) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If the rules for <literal> or for <unquoted interval string> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

ii) Otherwise,

1) If a <datetime value> does not conform to the natural rules for intervals according to the Gregorian calendar, then an exception condition is raised: *data exception—invalid interval format*.

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

c) If *SD* is interval and *TD* and *SD* have the same interval precision, then *TV* is *SV*.

d) If *SD* is interval and *TD* and *SD* have different interval precisions, then let *Q* be the least significant <datetime field> of *TD*.

i) Let *Y* be the result of converting *SV* to a scalar in units *Q* according to the natural rules for intervals as defined in the Gregorian calendar.

ii) Normalize *Y* to conform to the datetime qualifier "*P* TO *Q*" of *TD*. If this would result in loss of precision of the leading datetime field of *Y*, then an exception condition is raised: *data exception—interval field overflow*.

iii) *TV* is the value of *Y*.

17) If *TD* is an enumerated type, then

Case:

a) If *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

i) If *SV* is the same and an <enumeration name> defined within *TD*, then let *TV* be the <enumeration name> corresponding to that character string.

ii) Otherwise, an exception condition is raised: *data exception—invalid enumeration name*.

b) If *SD* is exact numeric, then

Case:

i) If *SV* is negative or *SV* is greater than the largest position number in *TD*, then an exception condition is raised: *data exception—invalid enumeration name*.

ii) Otherwise, *TV* is the <enumeration name> whose position number is *SV* in *TD*.

c) If *SD* is an enumerated type, then *TV* is the value of

CAST (CAST (*SV* AS INTEGER) AS *D*)

18) If *TD* is boolean and *SD* is character string, then *SV* is replaced by

TRIM (BOTH ' ' FROM *SV*)

Case:

a) If the rules for <literal> in Subclause 5.3, "<literal>", can be applied to *SV* to determine a valid value of the data type *TD*, then let *TV* be that value.

b) Otherwise, an exception condition is raised: *data exception—invalid character value for cast*.

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

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <cast specification>.

6.15 <value expression>

Function

Specify a value.

Format

```
<value expression> ::=
        <numeric value expression>
      | <string value expression>
      | <datetime value expression>
      | <interval value expression>
      | <enumerated value expression>
      | <boolean value expression>
```

ANSI Only--SQL3

```
      | <ADT value expression>
      | <OID value function>
```

```
      | <row value expression>
      | <distinct type value expression>
```

- 1 alternative deleted.

- 1 alternative deleted.

```
      | <collection value expression>
      | <operator expression>
```

ANSI Only--SQL3

```
<ADT value expression> ::= <value expression primary>
```

```
<distinct type value expression> ::=
      <value expression primary>

<collection value expression> ::=
        <set value constructor>
      | <multiset value constructor>
      | <list value constructor>

<value expression primary> ::=
        <unsigned value specification>
      | <item reference>
```

- 1 alternative deleted

```
      | <set function specification>
      | <table subquery>
      | <case expression>
      | <left paren> <value expression> <right paren>
      | <cast specification>
      | <subtype treatment>
```

Syntax Rules

1) The data type and null class of a <value expression> are the data type and null class of the <numeric value expression>, <string value expression>, <datetime value expression>, <interval value expression>, <enumerated value expression>, <boolean value expression>, <collection value expression>, <distinct type value expression>, <table value expression>, or <operator expression>, respectively.

2) The data type of a <collection value expression> is the collection type of the <set value constructor>, <multiset value constructor>, or <list value constructor> that it immediately contains.

3) The data type of a <distinct type value expression> is a distinct type.

4) If the data type of a <value expression primary> is character string, then the collating sequence and coercibility attribute of the <value expression primary> are the collating sequence and coercibility attribute of the <unsigned value specification>, <item reference>, <set function specification>, <scalar subquery>, <case expression>, <value expression>, or <cast specification> immediately contained in the <value expression primary>.

5) An <item reference> that is a <value expression primary> shall be a column reference.

   Note: "Column reference" is defined in Subclause 6.3, "<item reference>".

6) Let *C* be some column. Let *VE* be the <value expression>. *C* is an underlying column of *VE* if and only if *C* is identified by some column reference contained in *VE*.

ANSI Only–SQL3

7) The data type of the <value expression primary> of an <ADT value expression> shall be an abstract data type.

Access Rules

None.

General Rules

1) When a <value expression> *V* is evaluated for a row *R* of a table, each reference to a column of that table by a column reference *CR* directly contained in *V* is the value of that column in that row.

2) The value of a <collection value expression> is the value of the <set value constructor>, <multiset value constructor>, or <list value constructor> that it immediately contains.

3) If a <value expression primary> is a <scalar subquery> and the result of the <subquery> is empty, then the result of the <value expression primary> is the general null value.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <value expression> shall not be an <enumerated value expression>.

   b) A <value expression> shall not be a <boolean value expression>.

   c) A <value expression> shall not specify a <collection value expression>.

   d) A <value expression primary> that is a <table subquery> shall satisfy the Syntax Rules and General Rules for a <scalar subquery>.

   e) A <value expression> shall not be a <distinct type value expression>.

   f) A <value expression primary> shall not be a <subtype treatment>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <value expression> shall not be a <datetime value expression>.

   b) A <value expression> shall not be an <interval value expression>.

   c) A <value expression primary> shall not be a <case expression>.

   d) A <value expression primary> shall not be a <cast specification>.

   e) A <value expression primary> shall not be a <scalar subquery> except when the <value expression primary> is simply contained in a <value expression> that is simply contained in the second <row value constructor> of a <comparison predicate>.

6.16 <subtype treatment>

Function

Specifies a type.

Format

```
<subtype treatment> ::= TREAT ( <subtype operand> AS <target data type>

<subtype operand> ::= <value expression>

<target data type> ::=
         <domain name>
       | <abstract data type name>
```

Syntax Rules

1) The data type of the <value expression> shall be an abstract data type. Let *VT* be the abstract data type of the <value expression>.

2) The data type of the result of the <subtype treatment> is *DT*.

3) Case:

   a) If a <domain name> is specified, then let *DT* be the data type of the domain identified by <domain name>. *DT* shall be an abstract data type.

   b) Otherwise, let *DT* be the abstract data type identified by <abstract data type name>.

4) *VT* shall be a supertype of *DT*.

Access Rules

None.

General Rules

1) Let *V* be the value of the <value expression>.

2) If *DT* is a proper subtype of the most specific type of *V*, then an exception condition is raised: *invalid target type specification*.

   Note: “proper subtype” and “most specific type” are defined in Subclause 4.11.5, "Subtypes and supertypes for ADTs".

3) The value of the result of the <subtype treatment> is *V*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL Language shall contain no <subtype treatment>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.17 <numeric value expression>

Function

Specify a numeric value.

Format

```
<numeric value expression> ::=
         <term>
       | <numeric value expression> <plus sign> <term>
       | <numeric value expression> <minus sign> <term>

<term> ::=
         <factor>
       | <term> <asterisk> <factor>
       | <term> <solidus> <factor>

<factor> ::=
       [ <sign> ] <numeric primary>

<numeric primary> ::=
         <value expression primary>
       | <numeric value function>
```

Syntax Rules

1) Let *OP* be a <plus sign>, <minus sign>, <asterisk>, or <solidus>. If one or both operands of a <numeric value expression> that simply contains *OP* is an abstract data type, then:

   a) Let *OPND1* and *OPND2* be the first and second operands, respectively, of the <numeric value expression>.

   b) Let *OPID* be a <delimited identifier> whose <delimited identifier body> is *OP*.

   c) The <numeric value expression> is equivalent to the function invocation:

      OPID(*OPND1*,*OPND2*)

2) If a <factor> *F* contains a <sign> *S* and the data type of the <numeric primary> *NP* simply contained in *F* is an abstract data type, then:

   a) Let *OPID* be a <delimited identifier> whose <delimited identifier body> is *S*.

   b) The <factor> is equivalent to the function invocation:

      OPID(*NP*)

****Editor's Note****
Paper X3H2-94-244/SOU-097 noted that the data type of a <numeric primary> cannot be an ADT, but this Rule seems to depend on it. See Possible Problem **407** in the Editor's Notes.

3) If the data type of both operands of a dyadic arithmetic operator is exact numeric, then the data type of the result is exact numeric, with precision and scale determined as follows:

   a) Let *S1* and *S2* be the scale of the first and second operands respectively.

   b) The precision of the result of addition and subtraction is implementation-defined, and the scale is the maximum of *S1* and *S2*.

   c) The precision of the result of multiplication is implementation-defined, and the scale is *S1* + *S2*.

   d) The precision and scale of the result of division is implementation-defined.

4) If the data type of either operand of a dyadic arithmetic operator is approximate numeric, then the data type of the result is approximate numeric. The precision of the result is implementation-defined.

5) The data type of a <factor> is that of the immediately contained <numeric primary>.

6) The data type of a <numeric primary> shall be numeric.

7) If all the <numeric primary>s that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If the value of any <numeric primary> simply contained in a <numeric value expression> is a null value, then

   Case:

   a) If the result has the general null class, then the result of the <numeric value expression> is the general null value.

   b) Otherwise, the result of the <numeric value expression> is the minimum of the null values of the operands.

2) If the <numeric value expression> contains only a <numeric primary>, then the result of the <numeric value expression> is the value of the specified <numeric primary>.

3) The monadic arithmetic operators <plus sign> and <minus sign> (+ and −, respectively) specify monadic plus and monadic minus, respectively. Monadic plus does not change its operand. Monadic minus reverses the sign of its operand.

4) The dyadic arithmetic operators <plus sign>, <minus sign>, <asterisk>, and <solidus> (+, −, ∗, and /, respectively) specify addition, subtraction, multiplication, and division, respectively. If the value of a divisor is zero, then an exception condition is raised: *data exception—division by zero*.

5) If the type of the result of an arithmetic operation is exact numeric, then

Case:

a) If the operator is not division and the mathematical result of the operation is not exactly representable with the precision and scale of the result type, then an exception condition is raised: *data exception—numeric value out of range*.

b) If the operator is division and the approximate mathematical result of the operation represented with the precision and scale of the result type loses one or more leading significant digits after rounding or truncating if necessary, then an exception condition is raised: *data exception—numeric value out of range*. The choice of whether to round or truncate is implementation-defined.

6) If the type of the result of an arithmetic operation is approximate numeric and the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result type, then an exception condition is raised: *data exception—numeric value out of range*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

6.18 <string value expression>

Function

Specify a character string value or a bit string value.

Format

```
<string value expression> ::=
         <character value expression>
       | <bit value expression>
       | <blob value expression>

<character value expression> ::=
         <concatenation>
       | <character factor>

<concatenation> ::=
       <character value expression> <concatenation operator> <character factor>

<character factor> ::=
       <character primary> [ <collate clause> ]

<character primary> ::=
         <value expression primary>
       | <string value function>

<blob value expression> ::=
         <blob concatenation>
       | <blob factor>

<blob factor> ::= <blob primary>

<blob primary> ::=
         <value expression primary>
       | <string value function>

<blob concatenation> ::=
       <blob value expression> <concatenation operator> <blob factor>

<bit value expression> ::=
         <bit concatenation>
       | <bit factor>

<bit concatenation> ::=
       <bit value expression> <concatenation operator> <bit factor>

<bit factor> ::= <bit primary>

<bit primary> ::=
         <value expression primary>
       | <string value function>
```

Syntax Rules

1) If one or both operands of a <concatenation> is an abstract data type, then:

   a) Let *OPND1* and *OPND2* be the first and second operands, respectively, of the <concatenation>.

   b) Let *OPID* be a <delimited identifier> whose <delimited identifier body> is the <concatenation operator>.

   c) The <string value expression> is equivalent to the function invocation:

      OPID(*OPND1*,*OPND2*)

****Editor's Note****
Paper X3H2-94-244/SOU-097 noted that the data type of a <character factor> must be character string, but this Rule seems to accomodate one or both operands of a <concatenation> being an ADT. See Possible Problem **408** in the Editor's Notes.

2) The data type of a <character primary> shall be character string.

3) Character strings of different character repertoires shall not be mixed in a <character value expression>. The character repertoire of a <character value expression> is the character repertoire of its components.

4) Case:

   a) If <concatenation> is specified, then:

      Let *D1* be the data type of the <character value expression> and let *D2* be the data type of the <character factor>. Let *M* be the length in characters of *D1* plus the length in characters of *D2*. Let *VL* be the implementation-defined maximum length of a variable-length character string and let *FL* be the implementation-defined maximum length of a fixed-length character string.

      Case:

      i) If the data type of the <character value expression> or <character factor> is variable-length character string, then the data type of the <concatenation> is variable-length character string with maximum length equal to the lesser of *M* and *VL*.

      ii) If the data type of the <character value expression> and <character factor> is fixed-length character string, then *M* shall not be greater than *FL* and the data type of the <concatenation> is fixed-length character string with length *M*.

   b) Otherwise, the data type of the <character value expression> is the data type of the <character factor>.

5) Case:

   a) If <character factor> is specified,

      Case:

      i) If <collate clause> is specified, then the <character value expression> has the collating sequence given in <collate clause>, and has the *Explicit* coercibility attribute.

ii) Otherwise, if <value expression primary> or <string value function> are specified, then the collating sequence and coercibility attribute of the <character factor> are specified in Subclause 6.2, "<value specification> and <target specification>", and Subclause 6.10, "<string value function>", respectively.

b) If <concatenation> is specified, then the collating sequence and the coercibility attribute are determined as specified for dyadic operators in Subclause 4.2.3, "Rules determining collating sequence usage".

6) The data type of <blob primary> shall be binary string.

7) If <blob concatenation> is specified, then let *M* be the length in octets of the <blob value expression> plus the length in octets of the <blob factor> and let *VL* be the implementation-defined maximum length of a binary string. The data type of <blob concatenation> is binary string with maximum length equal to the lesser of *M* and *VL*.

8) The data type of a <bit primary> shall be bit string.

9) Case:

a) If <bit concatenation> is specified, then let *D1* be the data type of the <bit value expression>, let *D2* be the data type of the <bit factor>, let *M* be the length in bits of *D1* plus the length in bits of *D2*, let *VL* be the implementation-defined maximum length of a variable-length bit string, and let *FL* be the implementation-defined maximum length of a fixed-length bit string.

Case:

i) If the data type of the <bit value expression> or <bit factor> is variable-length bit string, then the data type of the <bit concatenation> is variable-length bit string with maximum length equal to the lesser of *M* and *VL*.

ii) If the data type of the <bit value expression> and <bit factor> is fixed-length bit string, then *M* shall not be greater than *FL* and the data type of the <bit concatenation> is fixed-length bit string with length *M*.

iii) Otherwise, the data type of a <bit value expression> is the data type of the <bit factor>.

10) If all the <character primary>s, <bit primary>s, or <blob primary>s contained in a <string value expression> that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If the value of any <character primary>, <bit primary>, or <blob primary> simply contained in a <string value expression> is a null value, then

Case:

a) If the result has the general null class, then the result of the <string value expression> is the general null value.

b) Otherwise, the result of the <string value expression> is the minimum of the null values of the operands.

2) If <concatenation> is specified, then let *S1* and *S2* be the result of the <character value expression> and <character factor>, respectively.

Case:

a) If either *S1* or *S2* is a null value, then the result of the <concatenation> is the general null value.

b) Otherwise, let *S* be the string consisting of *S1* followed by *S2* and let *M* be the length of *S*.

Case:

i) If the data type of either *S1* or *S2* is variable-length character string, then

Case:

1) If *M* is less than or equal to *VL*, then the result of the <concatenation> is *S* with length *M*.

2) If *M* is greater than *VL* and the right-most *M*−*VL* characters of *S* are all the <space> character, then the result of the <concatenation> is the first *VL* characters of *S* with length *VL*.

3) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

ii) If the data types of both *S1* and *S2* are fixed-length character string, then the result of the <concatenation> is *S*.

3) If <bit concatenation> is specified, then let *S1* and *S2* be the result of the <bit value expression> and <bit factor>, respectively.

Case:

a) If either *S1* or *S2* is a null value, then the result of the <bit concatenation> is the general null value.

b) Otherwise, let *S* be the string consisting of *S1* followed by *S2* and let *M* be the length in bits of *S*.

Case:

i) If the data type of either *S1* or *S2* is variable-length bit string, then

Case:

1) If *M* is less than or equal to *VL*, then the result of the <bit concatenation> is *S* with length *M*.

2) If *M* is greater than *VL* and the right-most *M*−*VL* bits of *S* are all 0-valued, then the result of the <bit concatenation> is the first *VL* bits of *S* with length *VL*.

3) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

ii) If the data types of both *S1* and *S2* are fixed-length bit string, then the result of the <bit concatenation> is *S*.

4) If <blob concatenation> is specified, then let *S1* and *S2* be the result of the <blob value expression> and <blob factor>, respectively.

Case:

a) If either *S1* or *S2* is a null value, then the result of the <blob concatenation> is the general null value.

b) Otherwise, let *S* be the string consisting of *S1* followed by *S2* and let *M* be the length in octets of *S*.

Case:

i) If *M* is less or equal to *VL*, then the result of the <blob concatenation> is *S* with length *M*.

ii) If *M* is greater than *VL* and the right-most $M - VL$ octets of *S* are all X'00', then the result of the <blob concatenation> is the first *VL* octets of *S* with length *VL*.

iii) Otherwise, an exception condition is raised: *data exception—string data, right truncation*.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <string value expression> shall not be a <blob value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

b) Conforming Intermediate SQL language shall contain no <bit value expression>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <character value expression> shall not be a <concatenation>.

6.19 <enumerated value expression>

Function

Specify an enumerated type value.

Format

```
<enumerated value expression> ::=
         <domain name> <left paren> <value expression> <right paren>
       | <enumerated primary>

<enumerated primary> ::=
         <value expression primary>
```

Syntax Rules

1) If <domain name> (<value expression>) is specified, then the data type of the <domain name> shall be an enumerated type and the data type of the <value expression> shall be exact numeric with a scale of 0. The data type of the <enumerated value expression> is the enumerated type <domain name>.

2) The data type of an <enumerated primary> shall be an enumerated type. The data type of the result is that enumerated type.

3) The result has the null class of the <enumerated primary>.

Access Rules

None.

General Rules

1) If the value of the <enumerated primary> simply contained in an <enumerated value expression> is a null value, then

   Case:

   a) If the result has the general null class, then the result of the <enumerated value expression> is the general null value.

   b) Otherwise it is that null value.

2) If <domain name>(<value expression>) is specified, then let *SV* be the result of the <value expression>.

   Case:

   a) If *SV* is a null value, then the result of the <enumerated value expression> is the general null value.

   b) If *SV* is negative or if *SV* is greater than the largest position number of <domain name>, then an exception condition is raised: *data exception—invalid enumeration value.*

c) Otherwise, the result of the <enumerated value expression> is the <enumeration literal> of <domain name> whose position number is *SV*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <enumerated value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.20 <datetime value expression>

Function

Specify a datetime value.

Format

```
<datetime value expression> ::=
         <datetime term>
       | <interval value expression> <plus sign> <datetime term>
       | <datetime value expression> <plus sign> <interval term>
       | <datetime value expression> <minus sign> <interval term>

<datetime term> ::=
         <datetime factor>

<datetime factor> ::=
         <datetime primary> [ <time zone> ]

<datetime primary> ::=
         <value expression primary>
       | <datetime value function>

<time zone> ::=
       AT <time zone specifier>

<time zone specifier> ::=
         LOCAL
       | TIME ZONE <interval primary>
```

Syntax Rules

1) The data type of a <datetime primary> shall be datetime.

2) Case:

   a) If the <datetime value expression> is a <datetime term>, then the precision of the result of the <datetime value expression> is the precision of the <datetime value function> or <value expression primary> that it simply contains.

   b) Otherwise, the precision of the result of the <datetime value expression> is the precision of the <datetime value expression> or <datetime term> that it simply contains.

3) If an <interval value expression> or <interval term> is specified, then the <interval value expression> or <interval term> shall only contain <datetime field>s that are contained within the <datetime value expression> or <datetime term>.

4) The data type of the <interval value expression> immediately contained in a <time zone specifier> shall be INTERVAL HOUR TO MINUTE.

5) Case:

   a) If the data type of the <datetime primary> is DATE, then <time zone> shall not be specified.

b) If the data type of the <datetime primary> is TIME or TIMESTAMP and <time zone> is not specified, then "AT LOCAL" is implicit.

6) If all the <datetime primary>s contained in a <datetime value expression> that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If the value of any <datetime primary>, <interval value expression>, <datetime value expression>, or <interval term> simply contained in a <datetime value expression> is a null value, then

Case:

a) If the result has the general null class, then the result of the <datetime value expression> is the general null value.

b) Otherwise, the result of the <datetime value expression> is the minimum of the null values of the operands.

2) If <time zone> is specified and the <interval primary> immediately contained in <time zone specifier> is null, then the result of the <datetime value expression> is the general null value.

3) If a <datetime value expression> immediately contains the operator <plus sign> or <minus sign>, then the result is effectively evaluated as follows:

a) Case:

i) If <datetime value expression> immediately contains the operator <plus sign> and the <interval value expression> or <interval term> is not negative, or if <datetime value expression> immediately contains the operator <minus sign> and the <interval term> is negative, then successive <datetime field>s of the <interval value expression> or <interval term> are added to the corresponding fields of the <datetime value expression> or <datetime term>.

ii) Otherwise, successive <datetime field>s of the <interval value expression> or <interval term> are subtracted from the corresponding fields of the <datetime value expression> or <datetime term>.

b) Arithmetic is performed so as to maintain the integrity of the datetime data type that is the result of the <datetime value expression>. This may involve carry from or to the immediately next more significant <datetime field>. If the data type of the <datetime value expression> is TIME, then arithmetic on the HOUR <datetime field> is undertaken modulo 24. If the <interval value expression> or <interval term> is a year-month interval, then the DAY field of the result is the same as the DAY field of the <datetime term> or <datetime value expression>.

c) If, after the preceding step, any <datetime field> of the result is outside the permissible range of values for the field or the result is invalid based on the natural rules for dates and times, then an exception condition is raised: *data exception—datetime field overflow*.

Note: For the permissible range of values for <datetime field>s, see Table 11, "Valid values for fields in datetime items".

4) If <time zone> is specified or implied, then:

a) If LOCAL is specified, then let *TZ* be the current default time zone displacement of the SQL-session. Otherwise, let *TZ* be the value of the <simple value specification> simply contained in the <time zone>.

b) If the value of the <interval primary> immediately contained in <time zone specifier> is less than INTERVAL –'12:59' or greater than INTERVAL +'13:00', then an exception condition is raised: *data exception—invalid time zone displacement value*.

c) Let *DV* be the value of the <datetime primary> directly contained in the <datetime value expression> expressed as a datetime normalized to UTC.

d) The value of the <datetime value expression> is calculated as:

$$DV - TZ$$

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <datetime value expression>.

6.21 <interval value expression>

Function

Specify an interval value.

Format

```
<interval value expression> ::=
         <interval term>
       | <interval value expression 1> <plus sign> <interval term 1>
       | <interval value expression 1> <minus sign> <interval term 1>
       | <left paren> <datetime value expression> <minus sign>
             <datetime term> <right paren> <interval qualifier>

<interval term> ::=
         <interval factor>
       | <interval term 2> <asterisk> <factor>
       | <interval term 2> <solidus> <factor>
       | <term> <asterisk> <interval factor>

<interval factor> ::=
       [ <sign> ] <interval primary>

<interval primary> ::=
         <value expression primary> [ <interval qualifier> ]

<interval value expression 1> ::= <interval value expression>

<interval term 1> ::= <interval term>

<interval term 2> ::= <interval term>
```

Syntax Rules

1) The data type of an <interval value expression> is interval. The data type of an <interval primary> shall be interval.

2) Case:

   a) If the <interval value expression> simply contains an <interval qualifier>, then the result shall contain the <datetime field>s specified in the <interval qualifier>.

   b) If the <interval value expression> is an <interval term>, then the result of an <interval value expression> contains the same <datetime field>s as the <interval primary>.

   c) If <interval term 1> is specified, then the result contains all the <datetime field>s that are contained within either <interval value expression 1> or <interval term 1>.

3) Case:

   a) If <interval term 1> is a year-month interval, then <interval value expression 1> shall be a year-month interval.

   b) If <interval term 1> is a day-time interval, then <interval value expression 1> shall be a day-time interval.

4) If <datetime value expression> is specified, then <datetime value expression> and <datetime term> shall be comparable.

5) If all the <interval primary>s, <datetime value expression>s, <datetime term>s, and <factor>s contained in an <interval value expression> that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) If an <interval term> specifies "<term> * <interval factor>", then let *T* and *F* be respectively the value of the <term> and the value of the <interval factor>. The result of the <interval term> is the result of *F* * *T*.

2) If the value of any <interval primary>, <datetime value expression>, <datetime term>, or <factor> that is simply contained in an <interval value expression> is a null value, then

   Case:

   a) If the result has the general null class, then the result of the <interval value expression> is the general null value.

   b) Otherwise, the result of the <interval value expression> is the minimum of the null values of the operands.

3) If the <sign> of an <interval factor> is <minus sign>, then the value of the <interval factor> is the negative of the value of the <interval primary>.

4) If <interval term 2> is specified, then:

   a) Let *X* be the value of <interval term 2> and let *Y* be the value of <factor>.

   b) Let *P* and *Q* be respectively the most significant and least significant <datetime field>s of <interval term 2>.

   c) Let *E* be an exact numeric result of the operation

      CAST (CAST (*X* AS INTERVAL *Q*) AS *E1*)

      where *E1* is an exact numeric data type of sufficient scale and precision so as to not lose significant digits.

   d) Let *OP* be the operator ∗ or / specified in the <interval value expression>.

   e) Let *I*, the result of the <interval value expression> expressed in terms of the <datetime field> *Q*, be the result of

      CAST ((*E OP Y*) AS INTERVAL *Q*).

   f) The result of the <interval value expression> is

      CAST (*I* AS INTERVAL *W*)

where *W* is an <interval qualifier> identifying the <datetime field>s *P* TO *Q*, but with <interval leading field precision> such that significant digits are not lost.

5) If <interval term 1> is specified, then let *P* and *Q* be respectively the most significant and least significant <datetime field>s in <interval term 1> and <interval value expression 1>, let *X* be the value of <interval value expression 1>, and let *Y* be the value of <interval term 1>.

   a) Let *A* be an exact numeric result of the operation

      CAST (CAST (*X* AS INTERVAL *Q*) AS *E1*)

      where *E1* is an exact numeric data type of sufficient scale and precision so as to not lose significant digits.

   b) Let *B* be an exact numeric result of the operation

      CAST (CAST (*Y* AS INTERVAL *Q*) AS *E2*)

      where *E2* is an exact numeric data type of sufficient scale and precision so as to not lose significant digits.

   c) Let *OP* be the operator + or − specified in the <interval value expression>.

   d) Let *I*, the result of the <interval value expression> expressed in terms of the <datetime field> *Q*, be the result of:

      CAST ((*A OP B*) AS INTERVAL *Q*)

   e) The result of the <interval value expression> is

      CAST (*I* AS INTERVAL *W*)

      where *W* is an <interval qualifier> identifying the <datetime field>s *P* TO *Q*, but with <interval leading field precision> such that significant digits are not lost.

6) If <datetime value expression> is specified, then let *Y* be the least significant <datetime field> specified by <interval qualifier>. Let *A* be the value represented by <datetime value expression> and let *B* be the value represented by <datetime term>. Evaluation of <interval value expression> proceeds as follows:

   a) *A* and *B* are converted to integer scalars *A2* and *B2* respectively in units *Y* as displacements from some implementation-dependent start datetime.

   b) The result is determined by effectively computing *A2*−*B2* and then converting the difference to an interval using an <interval qualifier> whose <end field> is *Y* and whose <start field> is sufficiently significant to avoid loss of significant digits. That interval is then converted to an interval using the specified <interval qualifier>, rounding or truncating if necessary. The choice of whether to round or truncate is implementation-defined.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <interval value expression>.

6.22 <operator expression>

Function

Invoke a routine with an operator.

Format

```
<operator expression> ::=
       <level 19 expression>

<level 19 expression>::=
         <level 19 monadic expression>
       | <level 19 infix expression>
       | <level 18 expression>

<level 19 monadic expression> ::=
         <level 19 prefix expression>
       | <level 19 postfix expression>

<level 19 prefix expression> ::=
         <level 19 prefix operator> <level 19 monadic expression>
       | <level 19 prefix operator> <level 18 expression>

<level 19 postfix expression> ::=
         <level 19 monadic expression> <level 19 postfix operator>
       | <level 18 expression> <level 19 postfix operator>

<level 19 infix expression> ::=
         <level 19 expression> <level 19 infix operator> <level 19 monadic expression>

<level 18 expression> ::=
         <level 17 expression>
       | <level 18 expression> <level 18 infix operator> <level 17 expression>

<level 17 expression> ::=
         <level 16 expression>
       | <level 17 expression> <level 17 infix operator> <level 16 expression>

<level 16 expression> ::=
         <level 15 expression>
       | <level 16 expression> <level 16 infix operator> <level 15 expression>

<level 15 expression> ::=
         <level 14 expression>
       | <level 15 expression> <level 15 infix operator> <level 14 expression>

<level 14 expression> ::=
         <level 13 expression>
       | <level 14 expression> <level 14 infix operator> <level 13 expression>

<level 13 expression> ::=
         <level 12 expression>
       | <level 13 expression> <level 13 infix operator> <level 12 expression>

<level 12 expression> ::=
         <level 11 expression>
       | <level 12 expression> <level 12 infix operator> <level 11 expression>
```

6.22 <operator expression>

```
<level 11 expression> ::=
        <level 10 expression>
      | <level 11 expression> <level 11 infix operator> <level 10 expression>

<level 10 expression> ::=
        <level 10 monadic expression>
      | <level 10 infix expression>
      | <level 9 expression>

<level 10 monadic expression> ::=
        <level 10 prefix expression>
      | <level 10 postfix expression>

<level 10 prefix expression> ::=
        <level 10 prefix operator> <level 10 monadic expression>
      | <level 10 prefix operator> <level 9 expression>

<level 10 postfix expression> ::=
        <level 10 monadic expression> <level 10 postfix operator>
      | <level 9 expression> <level 10 postfix operator>

<level 10 infix expression> ::=
        <level 10 expression> <level 10 infix operator> <level 10 monadic expression>

<level 9 expression> ::=
      <value expression>

<level 19 prefix operator> ::= !! See the Syntax Rules

<level 19 postfix operator> ::= !! See the Syntax Rules

<level 19 infix operator> ::= !! See the Syntax Rules

<level 18 infix operator> ::= !! See the Syntax Rules

<level 17 infix operator> ::= !! See the Syntax Rules

<level 16 infix operator> ::= !! See the Syntax Rules

<level 15 infix operator> ::= !! See the Syntax Rules

<level 14 infix operator> ::= !! See the Syntax Rules

<level 13 infix operator> ::= !! See the Syntax Rules

<level 12 infix operator> ::= !! See the Syntax Rules

<level 11 infix operator> ::= !! See the Syntax Rules

<level 10 infix operator> ::= !! See the Syntax Rules

<level 10 prefix operator> ::= !! See the Syntax Rules

<level 10 postfix operator> ::= !! See the Syntax Rules
```

Syntax Rules

1) Case:

   a) If <operator expression> is contained in a <schema definition> *SD*, then let *SN* be the <schema name> of *SD*.

   b) If <operator expression> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement>, or contained in a <direct SQL statement> that is invoked directly, then let *SN* be a <schema name> containing the default unqualified schema name and the default catalog name of the SQL-session.

   c) Otherwise, let *SN* fbe the <schema name> that is specified or implicit for the <module>.

2) The schema identified by *SN* shall contain an <operators definition> *OD*.

3) For *k* ranging from 10 to 19, a <level *k* infix operator), <level *k* prefix operator), or <level *k* postfix operator) is an <operator> that is contained in an <operator group> in *OD* whose <operator form> is INFIX, PREFIX, or POSTFIX, respectively, and whose <operator level> is *k*.

4) For *k* ranging from 11 to 18, and for *j* equal to $k-1$, a <level *k* expression> immediately containing a <level *k* infix operator> *OP*, a <level *k* expression> *OPND1*, and a <level *j* expression> *OPND2* is equivalent to the <routine invocation>:

   OP(*OPND1*, *OPND2*)

5) For *k* equal to 10 and *k* equal to 19, a <level *k* expression> immediately containing a <level *k* infix operator> *OP*, a <level *k* expression> *OPND1*, and a <level *j* monadic expression> *OPND2* is equivalent to the <routine invocation>:

   OP(*OPND1*, *OPND2*)

6) For *k* equal to 10 and *k* equal to 19, and for *j* equal to $k-1$, a <level *k* prefix expression> immediately containing a <level *k* prefix operator> *OP* and either a <level *k* monadic expression> *OPND* or a <level *j* expression> *OPND* is equivalent to the <routine invocation>:

   OP(*OPND*)

7) For *k* equal to 10 and *k* equal to 19, and for *j* equal to $k-1$, a <level *k* postfix expression> immediately containing a <level *k* postfix operator> *OP* and either a <level *k* postfix expression> *OPND* or a <level *j* expression> *OPND* is equivalent to the <routine invocation>:

   OP(*OPND*)

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain an <operator expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

- 1 Subclause deleted.

6.23 <boolean value expression>

Function

Specify a boolean value.

Format

```
<boolean value expression> ::=
         <boolean term>
       | <boolean value expression> OR <boolean term>

<boolean term> ::=
         <boolean factor>
       | <boolean term> AND <boolean factor>

<boolean factor> ::=
       [ NOT ] <boolean primary>

<boolean primary> ::=
         <predicate>
       | <value expression primary>
```

Syntax Rules

1) The data type of a <boolean primary> shall be boolean.

2) Case:

   a) If either operand of a dyadic boolean operator (AND or OR) has the general null class, then the result has null class of the other.

   b) Otherwise,

      Case:

      i) If both operands have the same null class, then the result has that null class.

      ii) Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) The result is derived by the application of the specified boolean operators (“AND”, “OR”, and “NOT”) to the results derived from each <boolean primary>. If boolean operators are not specified, then the result of the <boolean value expression> is the result of the specified <boolean primary>.

2) NOT (*true*) is *false* , NOT (*false*) is *true* , and NOT (*null*) is the same null value.

3) Table 13, "Truth table for the AND boolean", and Table 14, "Truth table for the OR boolean", specify the semantics of AND and OR, respectively.

4) Case:

   a) If only one operand is null and the result of the truth table specifies that the result is null, then

      Case:

      i) If the result has the general null class, its value is the general null value.

      ii) Otherwise, its value is the same null value.

   b) Where both operands are null and the result does not have the general null class, then

      Case:

      i) If the operator is AND, then the result is the minimum of such null values.

      ii) Otherwise the result is the maximum of such null values.

Table 13—Truth table for the AND boolean

AND	*true*	*false*	*null*
true	*true*	*false*	*null*
false	*false*	*false*	*false*
null	*null*	*false*	*null*

Table 14—Truth table for the OR boolean

OR	*true*	*false*	*null*
true	*true*	*true*	*true*
false	*true*	*false*	*null*
null	*true*	*null*	*null*

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <boolean value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7 Query expressions

7.1 <row value constructor>

Function

Specify
ANSI a value or list of
ISO an ordered set of
values to be constructed into a row or partial row.

Format

```
<row value constructor> ::=
         <row value constructor element>
       | [ ROW ] <left paren> <row value constructor list> <right paren>
       | <row subquery>

<row value constructor list> ::=
       <row value constructor element>
           [ { <comma> <row value constructor element> }... ]

<row value constructor element> ::=
         <value expression>
       | <null specification>
       | <default specification>

<null specification> ::=
       NULL [ <left paren> <null state> <right paren> ]

<default specification> ::=
       DEFAULT
```

Syntax Rules

1) If a <row value constructor> simply contains a <null specification> or a <default specification>, then either:

   a) the <row value constructor> shall be simply contained in a <query expression> that is simply contained in an <insert statement>; or

   b) the <row value constructor> shall be immediately contained in a <set clause>.

2) The data type of a <row value constructor element> that immediately contains <null specification> or <default specification> is the data type indicated in the column descriptor for the corresponding column in the explicit or implicit <insert column list> simply contained in the <insert statement>.

3) A <row value constructor element> immediately contained in a <row value constructor> shall not be a <value expression> of the form "<left paren> <value expression> <right paren>".

   Note: This Rule removes a syntactic ambiguity. A <row value constructor> of this form is permitted, but is parsed in the form "<left paren> <row value constructor list> <right paren>".

4) Let *RVC* be the <row value constructor>.

   Case:

   a) If *RVC* immediately contains a <row subquery>, then the data type of *RVC* is the data type of that <subquery>.

   b) Otherwise, the data type of *RVC* is a row type described by a sequence of (<field name>, data type) pairs, corresponding in order to each <row value constructor element> *X* simply contained in *RVC*. The data type is the data type of *X* and the <field name> is implementation-dependent and different from the <column name> name of any column or field, other than itself, of a table referenced by any <table reference> contained in the SQL-statement.

5) The degree of a <row value constructor> is the degree of its data type.

6) The degree of a <row value constructor> is the degree of its data type.

7) If <null state> is specified, then <null state> shall be contained in the null class of the corresponding object column in the containing
   ANSI <insert statement> or <set clause>.
   ISO <insert statement>.

Access Rules

None.

General Rules

1) The value of a <null specification> is a null value.

2) The value of a <default specification> is

ISO Only–SQL3

the default value indicated in the column descriptor for the corresponding column in the explicit or implicit <insert column list> simply contained in the <insert statement>.

ANSI Only—caused by ISO changes not yet considered by ANSI

determined according to the General Rules of Subclause 11.9, "<default clause>".

3) Case:

ANSI Only–SQL3

a) If the <row value constructor> immediately contains a <row value constructor element> *X*, then the result of the <row value constructor> is the value of *X*.

b) If a <row value constructor list> is specified, then the result of the <row value constructor> is a row of columns, the value of whose *i*-th column is the value of the *i*-th <row value constructor element> in the <row value constructor list>.

c) If the <row value constructor> is a <row subquery>, then:

   i) Let *R* be the result of the <row subquery> and let *D* be the degree of *R*.

   ii) If the cardinality of *R* is 0, then the result of the <row value constructor> is *D* general null values.

   iii) If the cardinality of *R* is 1, then the result of the <row value constructor> is *R*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <null specification> shall contain no <null state>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <row value constructor> that is not simply contained in a <table value constructor> or an <overlaps predicate> shall not contain more than one <row value constructor element>.

   b) A <row value constructor> shall not be a <row subquery>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <row value constructor element> shall not specify DEFAULT.

7.2 <row value expression>

Function

Specify a row value.

Format

```
<row value expression> ::=
         <value specification>
       | <row reference>
       | <row value constructor>
```

Syntax Rules

1) The data type of a <value specification> that is a <row value expression> shall be a row type.

Access Rules

None.

General Rules

1) A <row value expression> specifies the row value denoted by the <value specification>, <row reference>, or <row value constructor>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <row value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.3 <table value constructor>

Function

Specify a set of <row value expression>s to be constructed into a table.

Format

```
<table value constructor> ::=
       VALUES <table value constructor list>

<table value constructor list> ::=
       <row value expression> [ { <comma> <row value expression> }... ]
```

Syntax Rules

1) All <row value expression>s shall be of the same degree.

ANSI Only–SQL3

2) Let T_i be a table whose row type is the data type of the *i*-th <row value expression>.

3) The data type of the <table value constructor> is the data type of

   T_1 [UNION ALL T_2 [. . . UNION ALL T_n] . . .]

Access Rules

None.

General Rules

ANSI Only–SQL3

1) Let T_i be a table that contains one row that is the result of the *i*-th <row value expression>.

ISO Only—caused by ANSI changes not yet considered by ISO

2) Let T_i be a table whose *j*-th column has the same data type as the *j*-th <value expression> in the *i*-th <row value expression> and let T_i contain one row whose *j*-th column has the same value as the *j*-th <value expression> in the *i*-th <row value expression>.

3) The result of the <table value constructor> is the same as the result of

   T_1 [UNION ALL T_2 [. . . UNION ALL T_n] . . .]

4) If the result of any <row value expression> is a null value, then an exception condition is raised: *data exception—null row not permitted in table.*

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <table value constructor> shall contain exactly one <row value constructor> that shall be of the form "(<row value constructor list>)".

   b) A <table value constructor> shall be the <query expression> of an <insert statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.4 <set value constructor>

Function

Specify a collection of elements to be constructed into a set.

Format

```
<set value constructor> ::=
      SET <left paren> [ <collection list> ] <right paren>

<collection list> ::= <collection element> [ { <comma> <collection element> }... ]

<collection element> ::= <value expression>
```

Syntax Rules

1) Case:

   a) If <collection list> is omitted, then the <set value constructor> has the data type SET().

   b) Otherwise, the data type of the <set value constructor> is SET(*DT*), where *DT* is the data type determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the <collection element>s immediately contained in the <collection list> of this <set value constructor>.

2) Case:

   a) If no <collection list> is specified, then the <set value constructor> has the general null class.

   b) Otherwise, the null class of the <set value constructor> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the null classes of the <collection element>s immediately contained in the <collection list> of this <set value constructor>.

Access Rules

None.

General Rules

1) Case:

   a) If <collection list> is omitted, then the result of <set value constructor> is the empty set.

   b) Otherwise, the result of <set value constructor> is a set containing the value of each <collection element> immediately contained in the <collection list>, cast as the data type *DT*, with redundant duplicate elements eliminated.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <set value constructor>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.5 <multiset value constructor>

Function

Specify a collection of elements to be constructed into a multiset.

Format

```
<multiset value constructor> ::=
      MULTISET <left paren> [ <collection list> ] <right paren>
```

Syntax Rules

1) Case:

   a) If <collection list> is omitted, then the <multiset value constructor> has the data type MULTISET().

   b) Otherwise, the data type of the <multiset value constructor> is MULTISET(*DT*), where *DT* is the data type determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the <collection element>s immediately contained in the <collection list> of this <multiset value constructor>.

2) Case:

   a) If no <collection list> is specified, then the <multiset value constructor> has the general null class.

   b) Otherwise, the null class of the <multiset value constructor> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the null classes of the <collection element>s immediately contained in the <collection list> of this <multiset value constructor>.

Access Rules

None.

General Rules

1) Case:

   a) If <collection list> is omitted, then the result of <multiset value constructor> is the empty multiset.

   b) Otherwise, the result of <multiset value constructor> is a multiset containing the value of each <collection element> immediately contained in the <collection list>, cast as the data type *DT*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <multiset value constructor>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.6 <list value constructor>

Function

Specify a collection of elements to be constructed into a list.

Format

```
<list value constructor> ::=
      LIST <left paren> [ <collection list> ] <right paren>
```

Syntax Rules

1) Case:

   a) If <collection list> is omitted, then the <list value constructor> has the data type LIST( ).

   b) Otherwise, the data type of the <list value constructor> is LIST(*DT*), where *DT* is the data type determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the <collection element>s immediately contained in the <collection list> of this <list value constructor>.

2) Case:

   a) If no <collection list> is specified, then the <list value constructor> has the general null class.

   b) Otherwise, the null class of the <list value constructor> is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the null classes of the <collection element>s immediately contained in the <collection list> of this <list value constructor>.

Access Rules

None.

General Rules

1) Case:

   a) If <collection list> is omitted, then the result of <list value constructor> is the empty list.

   b) Otherwise, the result of <list value constructor> is a list whose *i*-th element is the value of the *i*-th <collection element> immediately contained in the <collection list>, cast as the data type *DT*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <list value constructor>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

7.7 <table expression>

Function

Specify a table or a grouped table.

Format

```
<table expression> ::=
       <from clause>
       [ <where clause> ]
       [ <group by clause> ]
       [ <having clause> ]
```

Syntax Rules

1) The result of a <table expression>is a derived table, whose
   ANSI data type
   ISO row type
   is the
   ANSI data type
   ISO row type
   of the result of the application of the last of the immediately contained clauses specified in the <table expression>.

2) Let *C* be some column. Let *TE* be the <table expression>. *C* is an underlying column of *TE* if and only if *C* is an underlying column of some column reference contained in *TE*.

Access Rules

None.

General Rules

1) If all optional clauses are omitted, then the result of the <table expression> is the same as the result of the <from clause>. Otherwise, each specified clause is applied to the result of the previously specified clause and the result of the <table expression> is the result of the application of the last specified clause.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the table identified in the <from clause> is a grouped view, then the <table expression> shall not contain a <where clause>, <group by clause>, or <having clause>.

7.8 <from clause>

Function

Specify a table derived from one or more named tables.

Format

```
<from clause> ::=
       FROM <table reference list>

<table reference list> ::=
       <table reference> [ { <comma> <table reference> }... ]
```

Syntax Rules

1) Case:

   a) If the <table reference list> contains a single <table reference> without an intervening <derived table> or <joined table>, then the descriptor of the result of the <table reference list> is the same as the descriptor of the table identified by that <table reference>. The
   ANSI data type
   ISO row type
   of the result of the <table reference list> is the
   ANSI data type
   ISO row type
   of the table identified by the <table reference>.

   b) If the <table reference list> contains more than one <table reference> with no intervening <derived table> or <joined table>, then the descriptors of the columns of the result of the <table reference list> are the descriptors of the columns of the tables identified by the <table reference>s, in the order in which the <table reference>s appear in the <table reference list> and in the order in which the columns are defined within each table. If each of the identified tables is a list table, then the result of the <table reference list> is also a list table. Otherwise, the result of the <table reference list> is a multiset table.

   ANSI Only–SQL3

   If each of the tables identified by the <table reference>s immediately contained in the <table reference list> is a list table, then the result of the <table reference list> is a list table; otherwise the result of the <table reference list> is a multiset table.

   The row type of the result of the <table reference list> is defined by the sequence of (<field name>, data type) pairs indicated by the sequence of column descriptors of the result of the <table reference list> taken in order.

2) The descriptor of the result of the <from clause> is the same as the descriptor of the result of the <table reference list>.

Access Rules

None.

General Rules

1) Case:

   a) If the <table reference list> contains a single <table reference> with no intervening <derived table> or <joined table>, then the result of the <table reference list> is the table identified by that <table reference>.

   b) If the <table reference list> contains more than one <table reference> without an intervening <derived table> or <joined table>, then the result of the <table reference list> is the extended Cartesian product of the tables identified by those <table reference>s.

      The extended Cartesian product, *CP*, is the multiset of all rows *R* such that *R* is the concatenation of a row from each of the identified tables in the order in which they are identified. The cardinality of *CP* is the product of the cardinalities of the identified tables. The ordinal position of a column in *CP* is $N+S$, where *N* is the ordinal position of that column in the identified table *T* from which it is derived and *S* is the sum of the degrees of the tables identified before *T* in the <table reference list>.

      ANSI Only–SQL3

      If each of the identified tables is a list table, then *CP* is a list table. If the result is a list table, then the order of the rows *R* is defined as the result of taking each row of the first identified table and concatenating it with the extended Cartesian product of the remaining identified tables in the order in which they were identified.

2) The result of the <from clause> is the result of the <table reference list>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the table identified by <table reference> is a grouped view, then the <from clause> shall contain exactly one <table reference>.

7.9 <joined table>

Function

Specify a table derived from a Cartesian product, inner or outer join, or union join.

Format

```
<joined table> ::=
         <cross join>
       | <qualified join>
       | <left paren> <joined table> <right paren>

<cross join> ::=
       <table reference> CROSS JOIN <table reference>

<qualified join> ::=
       <table reference> [ NATURAL ] [ <join type> ] JOIN
         <table reference> [ <join specification> ]

<join specification> ::=
         <join condition>
       | <named columns join>
       | <constraint join>

<join condition> ::= ON <search condition>

<named columns join> ::=
       USING <left paren> <join column list> <right paren>

<constraint join> ::=
       | USING PRIMARY KEY
       | USING FOREIGN KEY
       | USING CONSTRAINT <constraint name>

<join type> ::=
         INNER
       | <outer join type> [ OUTER ]
       | UNION

<outer join type> ::=
         LEFT
       | RIGHT
       | FULL

<join column list> ::= <column name list>
```

Syntax Rules

1) No <column name> contained in a <column name list> shall identify a column whose data type is a large object string.

2) Let TR_1 and TR_2 be the first and second <table reference>s of the <joined table>, respectively. Let T_1 and T_2 be the tables identified by TR_1 and TR_2, respectively. Let *TA* and *TB* be the correlation names of TR_1 and TR_2, respectively. Let *CP* be:

   SELECT * FROM TR_1, TR_2

3) If a <qualified join> is specified, then

   Case:

   a) If NATURAL is specified, then a <join specification> shall not be specified.

   b) If UNION is specified, then neither NATURAL nor a <join specification> shall be specified.

   c) Otherwise, a <join specification> shall be specified.

4) If a <qualified join> is specified and a <join type> is not specified, then INNER is implicit.

5) If a <qualified join> containing a <join condition> is specified, then:

   a) Each column reference directly contained in the <search condition> shall unambiguously reference a column of T_1 or T_2 or be an outer reference.

   b) If a <value expression> directly contained in the <search condition> is a <set function specification>, then the <joined table> shall be contained in a <having clause> or <select list> and the <set function specification> shall contain a column reference that is an outer reference.

   Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

6) If neither NATURAL is specified nor a <join specification> simply containing a <named columns join> is specified, then the descriptors of the columns of the result of the <joined table> are the same as the descriptors of the columns of *CP*.

7) If NATURAL is specified or if a <join specification> simply containing a <named columns join> is specified, then:

   a) If NATURAL is specified, then let *common column name* be a <column name> that is the <column name> of exactly one column of T_1 and the <column name> of exactly one column of T_2. T_1 shall not have any duplicate common column names and T_2 shall not have any duplicate common column names. Let *corresponding join columns* refer to all columns of T_1 and T_2 that have common column names, if any.

   b) If a <named columns join> is specified, then every <column name> in the <join column list> shall be the <column name> of exactly one column of T_1 and the <column name> of exactly one column of T_2. Let *common column name* be the name of such a column. Let *corresponding join columns* refer to the columns of T_1 and T_2 identified in the <join column list>.

   c) Let C_1 and C_2 be a pair of corresponding join columns contained in T_1 and T_2, respectively. C_1 and C_2 shall be comparable.

   d) Let *SLCC* be a <select list> of <derived column>s of the form

      COALESCE (*TA.C*, *TB.C*) AS *C*

      for every column *C* that is a corresponding join column, taken in order of their ordinal positions in T_1.

e) Let SLT_1 be a <select list> of those <column name>s of T_1 that are not corresponding join columns, taken in order of their ordinal positions in T_1, and let SLT_2 be a <select list> of those <column name>s of T_2 that are not corresponding join columns, taken in order of their ordinal positions in T_2.

f) The descriptors of the columns of the result of the <joined table> are the same as the descriptors of the columns of the result of

SELECT $SLCC$, SLT_1, SLT_2 FROM TR_1, TR_2

****Editor's Note****
I note that this General (sub)Rule appears to conflict with a later General Rule in this Subclause, as they both start off specifying "The descriptors of the columns of the result of the <joined table>...". Is there really a conflict here?

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

a) RIGHT, FULL, or UNION is specified, or

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

9) For every column CR of the result of the <joined table> that is not a corresponding join column and that corresponds to a column C_2 of T_2, CR is *possibly nullable* if any of the following conditions are true:

a) LEFT, FULL, or UNION is specified, or

b) INNER, RIGHT, or CROSS JOIN is specified or implicit and C_2 is possibly nullable.

10) For every column CR of the result of the <joined table> that is a corresponding join column and that corresponds to a column C_1 of T_1 and C_2 of T_2, CR is *possibly nullable* if any of the following conditions are true:

a) RIGHT, FULL, or UNION is specified and C_1 is possibly nullable, or

b) LEFT, FULL, or UNION is specified and C_2 is possibly nullable.

****Editor's Note****
Paper X3H2-90-023 significantly changed the SQL2 Rules for <joined table>, but did not make any changes to the SQL3 Rules. The SQL3 Rules must be updated correspondingly.

11) Case:

a) If NATURAL is specified, then:

i) Let *common column name* be a <column name> of a column of T_1 which is the same as a <column name> of some column of T_2.

ii) If there are no common column names, then *join columns* are not defined.

iii) If there is at least one common column name, then:

1) Let *join columns* refer to columns of T_1 and T_2 with identical common column names.

2) Let $JC1_i$ and $JC2_i$ be the i-th common column name of T_1, taken in the order of their ordinal position in T_1.

b) If PRIMARY KEY is specified, then:

i) The table descriptors for T_1 and T_2 shall each contain a unique constraint descriptor that specifies PRIMARY KEY.

ii) Let *join columns* refer to columns of T_1 and T_2 identified by the <unique column list>s in those <unique specification>s.

iii) Let $JC1_i$ be the name of the i-th column in the <unique column list> of T_1. Let $JC2_i$ be the name of the i-th column in the <unique column list> of T_2.

c) If FOREIGN KEY is specified, then:

i) There shall be exactly one referential constraint such that either T_1 is the referencing table and T_2 is the referenced table, or T_2 is the referencing table and T_1 is the referenced table.

Note: *referencing table* and *referenced table* are defined in Subclause 11.12, "<referential constraint definition>".

ii) Let *join columns* refer to the referencing columns and referenced columns of that referential constraint.

iii) If T_1 is the referencing table, then:

1) Let $JC1_i$ be the name of the i-th referencing column of the referential constraint.

2) Let $JC2_i$ be the name of the i-th referenced column of the referential constraint.

iv) If T_1 is the referenced table, then:

1) Let $JC1_i$ be the name of the i-th referenced column of the referential constraint.

2) Let $JC2_i$ be the name of the i-th referencing column of the referential constraint.

d) If CONSTRAINT is specified, then:

i) Case:

1) If the <table definition> for T_1 contains a <referential constraint definition> identified by <constraint name>, then the <referenced table> in that <referential constraint definition> shall refer to T_2.

2) Otherwise, the <table definition> for T_2 shall contain a <referential constraint definition> identified by <constraint name> whose <referential constraint definition> refers to T_1.

ii) Let *join columns* refer to columns of T_1 and T_2 identified by the <referencing columns> and <referenced table and columns> in that <referential constraint definition>.

iii) If T_1 is the referencing table, then:

1) Let $JC1_i$ be the name of the i-th column in the <referencing columns> in the <referential constraint definition>.

2) Let $JC2_i$ be the name of the *i*-th column in the <referenced table and columns> in the <referential constraint definition>.

iv) If T_1 is the referenced table, then:

1) Let $JC1_i$ be the name of the *i*-th column in the <referenced table and columns> in the <referential constraint definition>.

2) Let $JC2_i$ be the name of the *i*-th column in the <referencing columns> in the <referential constraint definition>.

e) Otherwise, *join columns* are not defined.

12) If NATURAL, PRIMARY KEY, FOREIGN KEY, or CONSTRAINT is specified, then

Case:

a) If join columns are defined, then:

i) The data type of the column identified by $JC1_i$ and the data type of the column identified by $JC2_i$ shall be comparable.

ii) Let *D1* be the domain of the column identified by $JC1_i$ and let *D2* be the domain of the column identified by $JC2_i$. If *D1* or *D2* is specified or implied, then both *D1* and *D2* shall specify or imply the same domain.

iii) Let *n* be the number of join columns.

iv) Let *SLCC* be a <select list> of *n* <value expression>s in which the *i*-th <value expression> is:

COALESCE ($T_1.JC1_i$, $T_2.JC2_i$) AS $JC1_i$

v) Let SLT_1 be a <select list> of *D1*−*n* <value expression>s in which the *i*-th <value expression> is the <column name> of the *i*-th column of T_1 that is not included in the join columns, taken in the order of their ordinal position in T_1.

vi) Let SLT_2 be a <select list> of *D2*−*n* <value expression>s in which the *i*-th <value expression> is the <column name> of the *i*-th column of T_2 that is not included in the join columns, taken in the order of their ordinal position in T_2.

vii) Let RVT_1 be a <row value constructor> of *n* <value expression>s where the *i*-th <value expression> is $T_1.JC1_i$.

viii) Let RVT_2 be a <row value constructor> of *n* <value expression>s where the *i*-th <value expression> is $T_2.JC2_i$.

ix) Let *JT* be the <join type>.

x) The <joined table> is equivalent to:

(SELECT *SLCC*, SLT_1, SLT_2 FROM T_1 *JT* JOIN T_2 ON RVT_1=RVT_2)

b) Otherwise, the <joined table> is equivalent to:

(SELECT ∗ FROM T_1, T_2)

13) The descriptors of the columns of the result of the <joined table> are the same as the descriptors of the columns of the tables identified by the <table reference>s in the <joined table> in the order in which the <table reference>s appear in the <joined table> and, within the table, in the order in which the columns are defined within that table. The data type of the rows of the <joined table> is the row type defined by the sequence of (<field name>, data type) pairs indicated by the sequence of column descriptors of the <joined table> taken in order.

****Editor's Note****

I note that the preceding General (sub)Rule appears to conflict with an earlier General Rule in this Subclause, as they both start off specifying "The descriptors of the columns of the result of the <joined table>...". Is there really a conflict here?

14) Case:

a) If PRIMARY KEY is specified, then:

i) Let *P1* be the <unique column list> of the <unique constraint definition> of T_1 that specifies PRIMARY KEY.

ii) The descriptor of the result of the <joined table> includes the constraint descriptor corresponding to the following <unique constraint definition>:

PRIMARY KEY (*P1*)

b) If FOREIGN KEY or CONSTRAINT is specified and the <table definition> of the referenced table contains a <unique constraint definition> that specifies PRIMARY KEY, then:

i) Let *P1* be the <unique column list> in that <unique constraint definition>.

ii) If *P1* is specified, the the descriptor of the result of the <joined table> includes the constraint descriptor corresponding to the following <unique constraint definition>:

PRIMARY KEY (*P1*)

c) Otherwise, the descriptor of the result of a <joined table> does not include a constraint descriptor corresponding to a <unique constraint definition>.

15) If PRIMARY KEY, FOREIGN KEY, or CONSTRAINT is specified and both T_1 and T_2 are inherently updatable, then the <joined table> is an inherently updatable table. Otherwise, the <joined table> is not an inherently updatable table.

Access Rules

None.

General Rules

1) Case:

a) If <join type> is UNION, then let *T* be the empty set.

b) If a <cross join> is specified, then let *T* be the multiset of rows of *CP*.

c) If a <join condition> is specified, then let *T* be the multiset of rows of *CP* for which the specified <search condition> is true.

d) If NATURAL is specified or <named columns join> is specified, then

Case:

i) If there are corresponding join columns, then let T be the multiset of rows of CP for which the corresponding join columns have equal values.

ii) Otherwise, let T be the multiset of rows of CP.

2) Let P_1 be the multiset of rows of T_1 for which there exists in T some row that is the concatenation of some row R_1 of T_1 and some row R_2 of T_2. Let P_2 be the multiset of rows of T_2 for which there exists in T some row that is the concatenation of some row R_1 of T_1 and some row R_2 of T_2.

3) Let U_1 be those rows of T_1 that are not in P_1 and let U_2 be those rows of T_2 that are not in P_2.

4) Let D_1 and D_2 be the degree of T_1 and T_2, respectively. Let X_1 be U_1 extended on the right with D_2 columns containing the general null value. Let X_2 be U_2 extended on the left with D_1 columns containing the general null value.

5) Let XN_1 and XN_2 be effective distinct names for X_1 and X_2, respectively. Let TN be an effective name for T.

Case:

a) If INNER or <cross join> is specified, then let S be the multiset of rows of T.

b) If LEFT is specified, then let S be the multiset of rows resulting from:

SELECT * FROM T
UNION ALL
SELECT * FROM X_1

c) If RIGHT is specified, then let S be the multiset of rows resulting from:

SELECT * FROM T
UNION ALL
SELECT * FROM X_2

d) If FULL is specified, then let S be the multiset of rows resulting from:

SELECT * FROM T
UNION ALL
SELECT * FROM X_1
UNION ALL
SELECT * FROM X_2

e) If UNION is specified, then let S be the multiset of rows resulting from:

SELECT * FROM X_1
UNION ALL
SELECT * FROM X_2

6) Let SN be an effective name of S.

Case:

a) If NATURAL is specified or a <named columns join> is specified, then the result of the <joined table> is the multiset of rows resulting from:

SELECT $SLCC$, SLT_1, SLT_2 FROM SN

b) Otherwise, the result of the <joined table> is S.

7) Case:

a) If PRIMARY KEY is specified, then:

i) If a row R is inserted in <joined table>, then:

1) Let V_j, V_1, and V_2 be the first n values, the next D_1-n values, and the last D_2-n values of R, respectively.

2) A candidate row CR_1 (CR_2) for T_1 (T_2) is effectively created whose join columns have the value V_j and columns not in join columns have the value V_1 (V_2).

3) If T_1 (T_2) does not contain a row whose value equals the value of CR_1 (CR_2), then CR_1 (CR_2) is inserted into T_1 (T_2).

ii) If a row R is deleted from <joined table>, then:

1) Let R_1 (R_2) be the row of T_1 (T_2) from which R is derived.

2) If R_1 (R_2) exists, then R_1 (R_2) is deleted from T_1 (T_2).

iii) If a row R of T is replaced in <joined table> by another row RR, then:

1) Let V_j, V_1, and V_2 be the first n values, the next D_1-n values, and the last D_2-n values of RR, respectively.

2) A candidate row CR_1 (CR_2) for T_1 (T_2) is effectively created whose join columns have the value V_j and columns not in join columns have the value V_1 (V_2).

3) Let R_1 (R_2) be the row of T_1 (T_2) from which R is derived.

4) If R_1 (R_2) exists, then R_1 (R_2) is replaced with CR_1 (CR_2).

5) If R_1 (R_2) does not exist and not all of the columns of CR_1 (CR_2) are a null value, then CR_1 (CR_2) is inserted into T_1 (T_2).

b) If FOREIGN KEY or CONSTRAINT is specified, then:

i) If a row R is inserted into the <joined table>, then:

1) Let V_j, V_1, and V_2 be the first n values, the next D_1-n values, and the last D_2-n values of R, respectively.

2) A candidate row CR_1 (CR_2) for T_1 (T_2) is effectively created whose join columns have the value V_j and columns not in join columns have the value V_1 (V_2).

3) If T_1 (T_2) is the referenced table and does not contain a row whose value equals the value of CR_1 (CR_2), then CR_1 (CR_2) is inserted into T_1 (T_2).

4) If T_1 (T_2) is the referencing table and contains a row NMR_1 (NMR_2) that is not one of the matching rows as defined in Subclause 11.12, "<referential constraint definition>", and the columns of T_1 (T_2) that are not in join columns have the value V_1 (V_2), and the <delete rule> of the <referential constraint definition> specifies SET NULL, then NMR_1 (NMR_2) is replaced by CR_1 (CR_2).

5) If T_1 (T_2) is the referencing table and each row in T_1 (T_2) is a matching row, then CR_1 (CR_2) is inserted into T_1 (T_2).

ii) If a row R is deleted from T, then:

1) Let R_1 (R_2) be the row in T_1 (T_2) from which R is derived.

2) If R_1 (R_2) does not exist, then delete R_2 (R_1) from T_2 (T_1).

3) If both R_1 and R_2 exist and T_1 (T_2) is the referencing table, then

Case:

A) If the <referential action> in the <referential constraint definition> specifies SET NULL, then the referencing columns of T_1 (T_2) are set to the general null value.

Note: *referencing columns* are defined in Subclause 11.12, "<referential constraint definition>".

B) Otherwise, R_1 (R_2) is deleted from T_1 (T_2).

iii) If a row R of T is replaced in <joined table> by another row RR, then:

1) Let V_j, V_1, and V_2 be the first n values, the next D_1-n values, and the last D_2-n values of RR, respectively.

2) A candidate row CR_1 (CR_2) for T_1 (T_2) is effectively created whose join columns have the value V_j and columns not in join columns have the value V_1 (V_2).

3) Let R_1 (R_2) be the row of T_1 (T_2) from which R is derived.

4) If R_1 (R_2) does not exist and not all of the columns of CR_1 (CR_2) are a null value, then CR_1 (CR_2) is inserted into T_1 (T_2).

5) If both R_1 and R_2 exist and T_1 (T_2) is the referenced table, then

Case:

A) If the row value of R_1 (R_2) is not equal to the row value of CR_1 (CR_2), then an exception condition is raised: *data exception—invalid update value*.

B) Otherwise, R_1 is replaced by CR_1 and R_2 is replaced by CR_2.

8) If UNION JOIN is specified, then:

a) Let D_1 be the degree of T_1 and let D_2 be the degree of T_2.

b) If a row R is inserted, then

Case:

i) If the first D_1 values of R are all a null value, and some of the last D_2 values of R are not a null value, then R is inserted into T_2.

ii) If the last D_2 values of R are all a null value, and some of the first D_1 values of R are not a null value, then R is inserted into T_1.

iii) Otherwise, an exception condition is raised: *data exception—invalid update value*.

c) If a row R is deleted from T, then

Case:

i) If R is derived from a row R_1 of T_1, then delete R_1 from T_1.

ii) If R is derived from a row R_2 of T_2, then delete R_2 from T_2.

d) If a row R of T_1 is replaced by some row RR, then

Case:

i) If R is derived from a row R_1 of T_1 and the last D_2 values of R are all a null value, then R_1 is replaced by RR.

ii) If R is derived from a row R_2 of T_2 and the first D_1 values of R are all a null value, then R_2 is replaced by RR.

iii) Otherwise, an exception condition is raised: *data exception—invalid update value.*

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <qualified join> shall contain no <constraint join>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall contain no <cross join>.

b) Conforming Intermediate SQL language shall not specify UNION JOIN.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <joined table>.

7.10 <where clause>

Function

Specify a table derived by the application of a <search condition> to the result of the preceding
ANSI <from clause> or <there is clause>.
ISO <from clause>.

Format

```
<where clause> ::= WHERE <search condition>
```

Syntax Rules

ANSI Only–SQL3

1) If the <where clause> is immediately contained in a <table expression>, then let *T* be the result of the preceding <from clause>; otherwise let *T* be the result of the <table reference> immediately contained in the <insert statement> containing the <where clause>. Each column reference directly contained in the <search condition> shall unambiguously reference a column of *T* or be an outer reference.

ISO Only—caused by ANSI changes not yet considered by ISO

2) Let *T* be the result of the preceding <from clause>. Each column reference directly contained in the <search condition> shall unambiguously reference a column of *T* or be an outer reference.

Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

3) If a <value expression> directly contained in the <search condition> is a <set function specification>, then the <where clause> shall be contained in a <having clause> or <select list> and every column reference in the <set function specification> shall be an outer reference.
Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

4) No column reference contained in a <subquery> in the <search condition> that references a column of *T* shall be specified in a <set function specification>.

5) The
ANSI data type
ISO row type
of the result of the <where clause> is the
ANSI data type
ISO row type
of *T*.

Access Rules

None.

General Rules

1) The <search condition> is applied to each row of *T*. The result of the <where clause> is a table of those rows of *T* for which the result of the <search condition> is true.

ANSI Only–SQL3

If *T* is a list table, then the result of the <where clause> retains the order of those rows of *T* that are in the result.

2) Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T*, then the reference is to the value of that column in the given row of *T*.

   Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <value expression> directly contained in the <search condition> shall not include a reference to a column that generally contains a <set function specification>.

7.11 <group by clause>

Function

Specify a grouped table derived by the application of the <group by clause> to the result of the previously specified clause.

Format

```
<group by clause> ::=
      GROUP BY <grouping column reference list>

<grouping column reference list> ::=
      <grouping column reference> [ { <comma> <grouping column reference> }... ]

<grouping column reference> ::=
      <item reference> [ <collate clause> ]
```

Syntax Rules

1) The data type of the column referenced by any <column reference> shall not be large object string.

2) If no <where clause> is specified, then let *T* be the result of the preceding <from clause>; otherwise, let *T* be the result of the preceding <where clause>.

3) Each <item reference> in the <group by clause> shall be a column reference and shall unambiguously reference a column of *T*. A column referenced in a <group by clause> is a grouping column.

   Note: "Column reference" is defined in Subclause 6.3, "<item reference>".

4) For every grouping column, if <collate clause> is specified, then the data type of the column reference shall be character string. The column descriptor of the corresponding column in the result has the collating sequence specified in <collate clause> and the coercibility attribute *Explicit*.

ANSI Only–SQL3

5) Let *DTT* be the data type of *T*. The data type of the result of the <group by clause> is:

   Case:

   a) If *DTT* is a list table type, then LIST(*DTT*).

   b) Otherwise, SET(*DTT*).

Access Rules

None.

General Rules

1) The result of the <group by clause> is a partitioning of *T* into a set of groups. The set is the minimum number of groups such that, for each grouping column of each group of more than one row, no two values of that grouping column are distinct.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is of <table type> LIST, then the result of the <group by clause> is an ordered set of groups, whose order is determined by the order in *T* of the first occurrence of a row from each group. Within each group, the relative order of the rows in *T* is retained.

2) Let *CR* be the column reference with <column name> *CN* identifying the grouping column. Every row of a given group contains equal values of *CN*. When a <search condition> or <value expression> is applied to a group, *CR* is a reference to the value of *CN*.

Note: See the General Rules of Subclause 8.2, "<comparison predicate>".

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

7.12 <having clause>

Function

Specify a grouped table derived by the elimination of groups that do not meet the <search condition>.

Format

```
<having clause> ::= HAVING <search condition>
```

Syntax Rules

1) The data type of the column referenced by any <column reference> directly contained in the <search condition> shall not be large object string.

2) Let *HC* be the <having clause>. Let *TE* be the <table expression> that immediately contains *HC*.

   Case:

   a) If *TE* immediately contains neither a <where clause> nor a <group by clause>, then let *T* be the descriptor of the table defined by the <from clause> *FC* immediately contained in *TE* and let *R* be the result of *FC*.

   b) If *TE* immediately contains a <where clause> *WC* but not a <group by clause>, then let *T* be the descriptor of the table defined by *WC* and let *R* be the result of *WC*.

   c) Otherwise, let *T* be the descriptor of the table defined by the <group by clause> *GBC* immediately contained in *TE* and let *R* be the result of *GBC*.

   If *TE* does not immediately contain a <group by clause>, then the table described by *T* has no grouping columns.

3) Each column reference directly contained in the <search condition> shall unambiguously reference a grouping column of *T* or be an outer reference.

   Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

4) Each column reference contained in a <subquery> in the <search condition> that references a column of *T* shall reference a grouping column of *T* or shall be specified within a <set function specification>.

5) The <having clause> is *possibly non-deterministic* if it contains a reference to a column *C* of *T* that has a data type of character string and:

   a) *C* is specified within a <set function specification> that specifies MIN or MAX, or

   b) *C* is a grouping column of *T*.

6) The
   ANSI data type
   ISO row type
   of the result of the <having clause> is the
   ANSI data type

ISO row type
of *T*.

Access Rules

None.

General Rules

1) If *TE* does not immediately contain a <group by clause> and *R* is not empty, then *R* consists of a single group.

2) The <search condition> is applied to each group of *R*. The result of the <having clause> is a grouped table of those groups of *R* for which the result of the <search condition> is true.

3) When the <search condition> is applied to a given group of *R*, that group is the argument or argument source of each <set function specification> directly contained in the <search condition>, unless the <column reference> in the <set function specification> is an outer reference.

4) Each <subquery> in the <search condition> is effectively executed for each group of *R* and the result used in the application of the <search condition> to the given group of *R*. If any evaluated <subquery> contains an outer reference to a column of *T*, then the reference is to the values of that column in the given group of *R*.

Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

7.13 <query specification>

Function

Specify a table derived from the result of a <table expression>.

Format

```
<query specification> ::=
       SELECT [ <set quantifier> ] <select list>
         <table expression>

<select list> ::=
         <asterisk>
       | <select sublist> [ { <comma> <select sublist> }... ]

<select sublist> ::=
         <derived column>
       | <item qualifier> <period> <asterisk>

<derived column> ::=
       <value expression> [ <as clause> ]

<as clause> ::= [ AS ] <column name>
```

Syntax Rules

1) Let *T* be the result of the <table expression>.

2) Let *TQS* be the table that is the result of a <query specification>.

3) The degree of the table specified by a <query specification> is equal to the cardinality of the <select list>.

- 1 Rule moved to Part 4

4) Case:

   a) If the <select list> "*" is simply contained in a <subquery> that is immediately contained in an <exists predicate>, then the <select list> is equivalent to a <value expression> that is an arbitrary <literal>.

   b) Otherwise, the <select list> "*" is equivalent to a <value expression> sequence in which each <value expression> is a column reference that references a column of *T* and each column of *T*
   ISO (excluding any column with <column name> OID)
   is referenced exactly once. The columns are referenced in the ascending sequence of their ordinal position within *T*.

5) If the <select sublist> "<item qualifier>.*" is specified, then let *Q* be the <item qualifier> of that <select sublist>. *Q* shall be a <table name or <correlation name> exposed by a <table reference> immediately contained in the <from clause> of *T*. Let *TQ* be the table associated with *Q*. That <select sublist> is equivalent to a <value expression> sequence in which each <value expression> is a column reference *CR* that references a column of *TQ* that is not a common column of a

ANSI <joined table>.
ISO <joined table> (excluding any column with <column name> OID).
Each column of *TQ* that is not a referenced common column shall be referenced exactly once. The columns shall be referenced in the ascending sequence of their ordinal positions within *TQ*.

Note: *common column of a <joined table>* is defined in Subclause 7.9, "<joined table>".

6) Let *C* be some column. Let *QS* be the <query specification>. Let DC_i, for *i* ranging from 1 to the number of <derived column>s inclusively, be the *i*-th <derived column> simply contained in the <select list> of *QS*. For all *i*, *C* is an underlying column of DC_i, and of any column reference that identifies DC_i, if and only if *C* is an underlying column of the <value expression> of DC_i, or *C* is an underlying column of the <table expression> immediately contained in *QS*.

7) Each column reference directly contained in each <value expression> and each column reference contained in a <set function specification> directly contained in each <value expression> shall unambiguously reference a column of *T*.

8) If *T* is a grouped table, then let *G* be the set defined by the <grouping column reference list>. In each <value expression>, each <column reference> that references a column of *T* shall reference some column *C* such that $G \mapsto C$, or shall be contained in a <set function specification>. If *T* is not a grouped table but some <value expression> contains a <set function specification> that contains a reference to a column of *T* or some <value expression> directly contains a <set function specification> that does not contain an outer reference, then in each <value expression>, each <column reference> that references a column of *T* shall be either contained in a <set function specification> or functionally dependent on the empty set.

****Editor's Note****
In discussions of X3H2-93-029/CBR-033, it has been noted that the use of Functional Dependencies in GROUP BY may have a problem when certain schema manipulation operations take place. See Possible Problem **271**.

9) Each column of *TQS* has a column descriptor that includes a data type descriptor that is the same as the data type descriptor of the <value expression> from which the column was derived.

10) Case:

a) If the *i*-th <derived column> in the <select list> specifies an <as clause> that contains a <column name> *CN*, then the <column name> of the *i*-th column of the result is *CN*.
ISO *CN* shall not be OID.

b) If the *i*-th <derived column> in the <select list> does not specify an <as clause> and the <value expression> of that <derived column> is a single column reference, then the <column name> of the *i*-th column of the result is the <column name> of the column designated by the column reference.

c) Otherwise, the <column name> of the *i*-th column of the <query specification> is implementation-dependent and different from the <column name> of any column, other than itself, of a table referenced by any <table reference> contained in the
ANSI SQL-statement.
ISO SQL-statement, and shall not be OID.

11) A column of *TQS* is *possibly nullable* if and only if it contains a column reference for a column *C* that is possibly nullable, an <indicator parameter>, a <subquery>, CAST NULL AS *X* (*X* represents a <data type> or a <domain name>), SYSTEM_USER, or a <set function specification> that does not contain COUNT.

12) Case:

    a) If a column *C* of *TQS* is derived from a <value expression> that comprises a column reference *CR*, and if *CR* identifies either a stored column that is inherently updatable or a virtual column with a set function, and no other column of *TQS* is derived from a <value expression> that comprises the same column reference, then *C* is inherently updatable.

    b) Otherwise, the column of *TQS* is not inherently updatable.

13) Let *TREF* be the <table reference>s that are simply contained in the <from clause> of the <table expression>. The *simply underlying table*s of the <query specification> are the tables identified by the <table name>s and <derived table>s contained in *TREF* without an intervening <derived table>.

14) A <query specification> *QS* is either *inherently updatable* or *not inherently updatable*. The updatability of *QS* is determined as follows:

    a) Let *UT1* denote some underlying table of *QS*, let *CKUT1* be some candidate key of *UT1*, let QSCOLS be the set of columns of *QS*, and let *QSCN1* be some exposed <table name> or exposed <correlation name> whose scope clause is *QS*.

    b) If some column *UC* of *UT1* has some *counterpart QC* in *QS*, then let *Q* be the explicit or implicit qualifier of *QC*'s column reference. *QC* is a *counterpart under Q* of *QC*.

       Note: *Counterpart* is defined in Subclause 4.22.1, "General rules and definitions".

    c) The truth-valued term "*Q*-to-*U* holds" is defined as follows:

       i) If and only if every member of *CKUT1* has some counterpart under *QSCN1* in *QSCOLS* (so that every row in *QS* corresponds to exactly one row in *UT1*, namely that row in *UT1* that has the same combined value in the columns of *CKUT1* as the row in *QS*), then *Q*-to-*U* holds, with respect to *QS* and *UT1*, under *QSCN1*.

       ii) If *Q*-to-*U* holds, with respect to *QS* and *UT1*, under *QSCN1*, then *QS* is said to be *partially inherently updatable* with respect to *UT1* under *QSCN1*.

    d) Let *CKQS* be some candidate key of *QS*, let *QSCN2* be some table or correlation name (possibly the same as *QSCN1*) whose scope clause is *QS*, let *UT2* be some underlying table of *QS*, possibly the same table as *UT1*, and let *UT2COLS* be the set of columns of *UT2*.

    e) The truth-valued term "*U*-to-*Q* holds" is defined as follows:

       i) If and only if every member of *CKQS* is a counterpart under *QSCN2* of some member of *UT2COLS* (so that every row in *UT2* corresponds to exactly one row in *QS*, namely that row in *QS* that has the same combined value in the columns of *CKQS* as the row in *UT2*), then *U*-to-*Q* holds, with respect to *UT2* and *QS*, under *QSCN2*.

       ii) If *Q*-to-*U* holds and *U*-to-*Q* holds, with respect to *QS* and *UT2*, under *QSCN2*, then *QS* is said to be *fully inherently updatable* with respect to *UT2* under *QSCN2*.

f) *QS* is inherently updatable if and only if:

i) every column of *QS* has a counterpart, under some qualifier, in some underlying table with respect to which *QS* is partially inherently updatable under that qualifier, and

ii) there is some underlying table with respect to which *QS* is fully inherently updatable under some qualifier.

15) A <query specification> is *possibly non-deterministic* if any of the following conditions are true:

a) The <set quantifier> DISTINCT is specified and one of the columns of *T* has a data type of character string; or

b) The <query specification> directly contains a <having clause> that is possibly non-deterministic; or

c) The <select list> contains a reference to a column *C* of *T* that has a data type of character string and either

i) *C* is specified with a <set function specification> that specifies MIN or MAX, or

ii) *C* is a grouping column of *T*.

ANSI Only–SQL3

16) Case:

a) If the <set quantifier> DISTINCT is specified, then *TQS* is a set table.

b) If the <set quantifier> DISTINCT is not specified and *T* is a list table, then *TQS* is a list table.

c) Otherwise, *TQS* is a multiset table.

17) The row type of *TQS* is defined by the sequence of (<field name>, data type) pairs indicated by the sequence of column descriptors of *TQS* taken in order.

Access Rules

None.

General Rules

1) Case:

a) If *T* is not a grouped table, then

Case:

i) If the <select list> contains a <set function specification> that contains a reference to a column of *T* or directly contains a <set function specification> that does not contain an outer reference, then *T* is the argument or argument source of each such <set function

specification> and the result of the <query specification> is a table consisting of 1 row. The *i*-th value of the row is the value specified by the *i*-th <value expression>.

ii) If the <select list> does not include a <set function specification> that contains a reference to *T*, then each <value expression> is applied to each row of *T* yielding a table of *M* rows, where *M* is the cardinality of *T*. The *i*-th column of the table contains the values derived by the evaluation of the *i*-th <value expression>.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a list table, then the table yielded is a list table with its rows in the same order as those of *T* from which they were produced.

Case:

1) If the <set quantifier> DISTINCT is not specified, then the result of the <query specification> is the table.

2) If the <set quantifier> DISTINCT is specified, then the result of the <query specification> is the table derived from that table by the elimination of any redundant duplicate rows.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a list table, then the order of the remaining rows is retained in the result.

b) If *T* is a grouped table, then

Case:

i) If *T* has 0 groups, then the result of the <query specification> is an empty table.

ii) If *T* has one or more groups, then each <value expression> is applied to each group of *T* yielding a table of *M* rows, where *M* is the number of groups in *T*. The *i*-th column of the table contains the values derived by the evaluation of the *i*-th <value expression>. When a <value expression> is applied to a given group of *T*, that group is the argument or argument source of each <set function specification> in the <value expression>.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a grouped list table, then the table yielded is a list table with its rows in the same order as the groups of *T* from which they were produced.

Case:

1) If the <set quantifier> DISTINCT is not specified, then the result of the <query specification> is the table.

2) If the <set quantifier> DISTINCT is specified, then the result of the <query specification> is the table derived from *T* by the elimination of any redundant duplicate rows.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a list table, the order of the remaining rows is retained in the result.

2) Let *UQS* be an inherently updatable <query specification>, let UCN_1, UCN_2, . . . ,UCN_n be those exposed <table name>s and exposed <correlation name>s whose scope clause is *UQS*, and let UT_1, UT_2, . . . ,UT_m be those underlying tables of *UQS* with respect to which *UQS* is partially inherently updatable.

   If *n*>1, then updating of *UQS* is subject to the constraints that would be implied by WITH CASCADED CHECK OPTION, were *UQS* the <query expression> of a <view definition>.

3) If a row, *UQS_ROW*, is inserted into *UQS*, then:

   a) If *UQS_ROW* IN *UQS* is true, then an exception condition is raised: *data exception—row already exists*.

   b) If two or more distinct columns in *UQS* have the same counterpart, under the same UCN_i, in UT_j, and *UQS_ROW* does not have the same value in each of those columns, then an exception condition is raised: *data exception—invalid update value*.

   c) For each underlying table UT_j, let $UTROW_j$ be the row constructed from the *UQS_ROW* values in columns that have counterparts in UT_j under UCN_i.

      Case:

      i) If $UTROW_j$ IN UT_j is true, then do nothing;

      ii) If there is no row in UT_j that matches $UTROW_j$ in all columns, and *UQS* is not fully inherently updatable with respect to UT_j under UCN_i, then an exception condition is raised: *data exception—invalid update value*.

      iii) Otherwise, $UTROW_j$ is expanded as necessary with default values for any missing columns, and an attempt is made to insert the resulting row into UT_j.

4) If a row *UQS_OLD* in *UQS* is updated, then let *UQS_NEW* be the row in UQS that would result from the update.

   For every column of *UQS*, if neither *UQS_NEW* has the same value in that column as *UQS_OLD*, nor does that column have a counterpart in some leaf generally underlying table of *UQS* with respect to which *UQS* is fully inherently updatable under some <correlation name> or exposed <table name>, then an exception condition is raised: *data exception—invalid update value*.

   For every *j*, *m*>=*j*>=1, if, for some *i*, UCN_i is a qualifier such that *UQS* is fully inherently updatable with respect to UT_j under UCN_i, then:

   a) Let $UTOLD_j$ be the row in UT_j that corresponds to *UQS_OLD*, and let $UTNEW_j$ be the row constructed from the *UQS_NEW* values in columns that have counterparts in UT_j under UCN_i, expanded as necessary with values from $UTOLD_j$ for the remaining columns of UT_j.

b) $UTOLD_j$ is replaced by $UTNEW_j$.

5) If a row *UQS_ROW* is deleted from *UQS*, then, for all *j*, m>=j>=1, if, for some *i*, UCN_i is a qualifier such that *UQS* is fully inherently updatable with respect to UT_j under UCN_i, then:

a) Let $UTROW_j$ be the row in UT_j that corresponds to *UQS_ROW*.

b) $UTROW_j$ is
ANSI marked for deletion
ISO deleted
from UT_j.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A query specification is not inherently updatable if a column reference appears more than once in a <select list> or if the <value expression> of a <derived column> is not a column reference.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) The <set quantifier> DISTINCT shall not be specified more than once in a <query specification>, excluding any <subquery> of that <query specification>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <query specification> is not inherently updatable if the <where clause> of the <table expression> contains a <subquery>.

b) A <select sublist> shall be a <derived column>.

c) If the <table expression> of the <query specification> is a grouped view, then the <select list> shall not contain a <set function specification>.

7.14 <query expression>

Function

Specify a table.

Format

ANSI Only---caused by ISO changes not yet considered by ANSI

```
<query expression> ::=
       <possibly updatable query expression> [ <trigger definition>... ]

<possibly updatable query expression> ::=
         <non-join query expression>
       | <joined table>
```

ISO Only--SQL3

```
<query expression> ::=
       [ <with clause> ] <query expression body>

<with clause > ::= WITH <with list>

<with list> ::=
       <with list element> [ { <comma> <with list element> }... ]

<with list element> ::=
       <query name>
         [ <left paren> <with column list> <right paren> ]
         AS <left paren> <query expression> <right paren>

<with column list> ::= <column name list>

<query expression body> ::=
         <non-join query expression>
       | <joined table>
```

```
<non-join query expression> ::=
         <non-join query term>
       | <query expression> UNION  [ ALL ] [ <corresponding spec> ] <query term>
       | <query expression> EXCEPT [ ALL ] [ <corresponding spec> ] <query term>

<query term> ::=
         <non-join query term>
       | <joined table>

<non-join query term> ::=
         <non-join query primary>
       | <query term> INTERSECT [ ALL ] [ <corresponding spec> ] <query primary>
       | <recursive union>

<query primary> ::=
```

```
        <non-join query primary>
      | <joined table>

<non-join query primary> ::=
        <simple table>
      | <left paren> <non-join query expression> <right paren>

<simple table> ::=
        <query specification>
      | <table value constructor>
      | <explicit table>
      | <collection expression>
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
<explicit table> ::= <table type> <table name>
```

ISO Only--SQL3

```
<explicit table> ::= TABLE <table name>
```

```
<corresponding spec> ::=
      CORRESPONDING [ BY <left paren> <corresponding column list> <right paren> ]

<corresponding column list> ::= <column name list>

<collection expression> ::=
      <value expression>
```

Syntax Rules

ISO Only–SQL3

1) If <with clause> is specified, then:

   a) Let n be the number of <with list element>s and let WLE_i and WLE_j be the i-th and j-th <with list element>s for every (i,j) with i ranging from 1 to n and j ranging from $i+1$ to n. WLE_i shall not immediatelay contain the <query name> immediately contained in WLE_j.

   b) For all i between 1 and n, the scope of the <query name> WQN immediately contained in WLE_i is the <query expression> immediately contained in every <with list element> WLE_k, where k ranges from $i+1$ to n, and the <query expression body> immediately contained in <query expression>.

c) For every <with list element> *WLE*, let *WQE* be the <query expression> specified by *WLE* and let *WQT* be the table defined by *WQE*.

i) If any two columns of *WQT* have the same name, then *WLE* shall specify a <with column list>. If *WLE* specifies a <with column list> *WCL*, then:

1) The same <column name> shall not be specified more than once in *WCL*.

2) The number of <column name>s in *WCL* shall be the same as the degree of *WQT*.

ii) No column in *WQT* shall have a coercibility attribute of *No collating sequence*.

2) Let *T* be the table specified by the <query expression>.

3) If <routine invocation> is specified in the <user-defined updatability clause>, then the subject routine shall be a function.

ANSI Only—caused by ISO changes not yet considered by ANSI

4) If the <table type> contained in <explicit table> is not TABLE, then it shall be the same as that in the descriptor for the table identified by the <table name> contained in <explicit table>.

5) The <explicit table>

TABLE <table name>

is equivalent to the <query expression>

ANSI Only—caused by ISO changes not yet considered by ANSI

(SELECT ROW *T* FROM <table name> *T*)

ISO Only—caused by ANSI changes not yet considered by ISO

(SELECT * FROM <table name>)

6) Let *set operator* be UNION [ALL], EXCEPT [ALL], or INTERSECT [ALL].

7) *T* is an inherently updatable table and the <query expression> is inherently updatable if and only if it simply contains a <query expression> *QE* or a <query specification> *QS* and:

a) the <query expression> contains *QE* or *QS* without an intervening <non-join query expression> that specified UNION or EXCEPT;

b) the <query expression> contains *QE* or *QS* without an intervening <non-join query term> that specifies INTERSECT; and

c) *QE* or *QS* is inherently updatable.

ANSI Only—caused by ISO changes not yet considered by ANSI

---

8) *T* is a *trigger-mutable table* and the <query expression> is *trigger-mutable* if and only if the <query expression> immediately contains each of the following:

a) A <trigger definition> containing <trigger action time> INSTEAD OF and <trigger event> INSERT.

b) A <trigger definition> containing <trigger action time> INSTEAD OF and <trigger event> DELETE.

c) One or more <trigger definition>s containing <trigger action time> INSTEAD OF and <trigger event> UPDATE, where each column of the <query expression> is identified in one of the <trigger column list>s.

9) A trigger-mutable table is effectively a base table with respect to updatability rules.

10) *T* is an *updatable table* and the <query expression> is *updatable* if and only if it is inherently updatable or it is trigger-mutable or both.

11) The data type of the <value expression> immediately contained in <collection expression> shall be a <collection type>.

---

12) Case:

a) If a <simple table> is a <query specification>, then the column descriptor of the *i*-th column of the <simple table> is the same as the column descriptor of the *i*-th column of the <query specification>.

b) If a <simple table> is an <explicit table>, then the column descriptor of the *i*-th column of the <simple table> is the same as the column descriptor of the *i*-th column of the table identified by the <table name> contained in the <explicit table>.

c) If a <simple table> is a <collection expression>, then the <simple table> has one column of that data type and the name of that column is implementation-dependent and different from the <column name> of any column, other than itself, of a table referenced by any <table reference> contained in the
ANSI SQL-statement.
ISO SQL-statement and the <column name> is not OID.

d) Otherwise, the column descriptor of the *i*-th column of the <simple table> is the same as the column descriptor of the *i*-th column of the <table value constructor>, except that the <column name> is implementation-dependent and different from the <column name> of any column, other than itself, of a table referenced by any <table reference> contained in the
ANSI SQL-statement.

ISO SQL-statement and the <column name> is not OID.

13) Case:

a) If a <non-join query primary> is a <simple table>, then the column descriptor of the *i*-th column of the <non-join query primary> is the same as the column descriptor of the *i*-th column of the <simple table>.

b) Otherwise, the column descriptor of the *i*-th column of the <non-join query primary> is the same as the column descriptor of the *i*-th column of the <non-join query expression>.

14) Case:

a) If a <query primary> is a <non-join query primary>, then the column descriptor of the *i*-th column of the <query primary> is the same as the column descriptor of the *i*-th column of the <non-join query primary>.

b) Otherwise, the column descriptor of the *i*-th column of the <query primary> is the same as the column descriptor of the *i*-th column of the <joined table>.

15) If a set operator is specified in a <non-join query term> or a <non-join query expression>, then let *T1*, *T2*, and *TR* be respectively the first operand, the second operand, and the result of the <non-join query term> or <non-join query expression>. Let *TN1* and *TN2* be the effective names for *T1* and *T2*, respectively.

16) If a set operator is specified in a <non-join query term> or a <non-join query expression>, then let *OP* be the set operator.

Case:

a) If CORRESPONDING is specified, then:

i) Within the columns of *T1*, the same <column name> shall not be specified more than once and within the columns of *T2*, the same <column name> shall not be specified more than once.

ii) At least one column of *T1* shall have a <column name> that is the <column name> of some column of *T2*.

iii) Case:

1) If <corresponding column list> is not specified, then let *SL* be a <select list> of those <column name>s that are <column name>s of both *T1* and *T2* in the order that those <column name>s appear in *T1*.

2) If <corresponding column list> is specified, then let *SL* be a <select list> of those <column name>s explicitly appearing in the <corresponding column list> in the order that these <column name>s appear in the <corresponding column list>. Every <column name> in the <corresponding column list> shall be a <column name> of both *T1* and *T2*.

iv) The <non-join query term> or <non-join query expression> is equivalent to:

(SELECT *SL* FROM *TN1*) *OP* (SELECT *SL* FROM *TN2*)

b) If CORRESPONDING is not specified, then *T1* and *T2* shall be of the same degree.

17) Case:

a) If the <non-join query term> is a <non-join query primary>, then the column descriptor of the *i*-th column of the <non-join query term> is the same as the column descriptor of the *i*-th column of the <non-join query primary>.

b) If the <non-join query term> is a <recursive union>, then the column name of the *i*-th column of the <non-join query term> is that of the *i*-th column of the <recursive union>. The data type of the *i*-th column of the <non-join query term> is the data type of the *i*-th column of the <recursive union>.

c) Otherwise,

i) Case:

1) Let *C* be the <column name> of the *i*-th column of *T1*. If the <column name> of the *i*-th column of *T2* is *C*, then the <column name> of the *i*-th column of *TR* is *C*.

2) Otherwise, the <column name> of the *i*-th column of *TR* is implementation-dependent and different from the <column name> of any column, other than itself, of any table referenced by any <table reference> contained in the SQL-statement.

ii) The data type and null class of the *i*-th column of *TR* is determined by applying Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the *i*-th column of *T1* and the *i*-th column of *T2*. If the *i*-th columns of either *T1* or *T2* are known not nullable, then the *i*-th column of *TR* is known not nullable; otherwise, the *i*-th column of *TR* is possibly nullable.

18) Case:

a) If a <query term> is a <non-join query term>, then the column descriptor of the *i*-th column of the <query term> is the same as the column descriptor of the *i*-th column of the <non-join query term>.

b) Otherwise, the column descriptor of the *i*-th column of the <query term> is the same as the column descriptor of the *i*-th column of the <joined table>.

19) Case:

a) If a <non-join query expression> is a <non-join query term>, then the column descriptor of the *i*-th column of the <non-join query expression> is the same as the column descriptor of the *i*-th column of the <non-join query term>.

b) Otherwise,

i) Case:

1) Let *C* be the <column name> of the *i*-th column of *T1*. If the <column name> of the *i*-th column of *T2* is *C*, then the <column name> of the *i*-th column of *TR* is *C*.

2) Otherwise, the <column name> of the *i*-th column of *TR* is implementation-dependent and different from the <column name> of any column, other than itself, of any table referenced by any <table reference> contained in the SQL-statement.

ii) The data type and null class of the *i*-th column of *TR* is determined by applying the Syntax Rules of Subclause 9.3, "Set operation result data types and nullabilities", to the data types of the *i*-th column of *T1* and the *i*-th column of *T2*.

Case:

1) If the <non-join query expression> immediately contains EXCEPT, then if the *i*-th column of *T1* is known not nullable, then the *i*-th column of *TR* is known not nullable; otherwise, the *i*-th column of *TR* is possibly nullable.

2) Otherwise, if the *i*-th columns of both *T1* and *T2* are known not nullable, then the *i*-th column of *TR* is known not nullable; otherwise, the *i*-th column of *TR* is possibly nullable.

20) Case:

a) If a <query expression> is a <non-join query expression>, then the column descriptor of the *i*-th column of the <query expression> is the same as the column descriptor of the *i*-th column of the <non-join query expression>.

b) Otherwise, the column descriptor of the *i*-th column of the <query expression> is the same as the column descriptor of the *i*-th column of the <joined table>.

21) The *simply underlying tables* of a <query expression> are the tables identified by those <table name>s, <query specification>s, and <derived table>s contained in the <query expression> without an intervening <derived table> or an intervening <join condition>.

22) A <query expression> is *possibly non-deterministic* if

a) it contains a set operator UNION and ALL is not specified, or if it contains EXCEPT or INTERSECT; and

b) the first or second operand contains a column that has a data type of character string.

23) The *underlying columns* of each column of *QE* and of *QE* itself are defined as follows:

a) A column of a <table value constructor> has no underlying columns.

b) The underlying columns of every *i*-th column of a <simple table> *ST* are the underlying columns of the *i*-th column of the table immediately contained in *ST*.

c) If no set operator is specified, then the underlying columns of every *i*-th column of *QE* are the underlying columns of the *i*-th column of the <simple table> simply contained in *QE*.

d) If a set operator is specified, then the underlying columns of every *i*-th column of *QE* are the underlying columns of the *i*-th column of *T1* and those of the *i*-th column of *T2*.

e) Let *C* be some column. *C* is an underlying column of *QE* if and only if *C* is an underlying column of some column of *QE*.

ANSI Only–SQL3

24) Case:

a) If <query expression> specifies a set operator that specifies ALL, then the result of <query expression> is a multiset table.

b) If <query expression> specifies a set operator and does not specify ALL, then the result of <query expression> is a set table.

c) If no set operator is specified, then the result of <query expression> is the result of the specified <simple table> or <joined table>.

---

25) Let *PUQE* be the <possibly inherently updatable query expression> immediately contained in <query expression>.

The row type of the result of <query expression> is the row type of the <non-join query expression> or <joined table> immediately contained in *PUQE*.

26) The row type of a <non-join query expression> *NJQE* is determined as follows.

Case:

a) If *NJQE* immediately contains <non-join query term> *NJQT*, then the row type of *NJQE* is that of *NJQT*.

b) Otherwise:

Case:

i) If no <corresponding spec> is specified and the <query expression> and <query term> immediately contained in *NJQE* have the same row type, then the row type of *NJQE* is that row type.

ii) Otherwise, the row type of *NJQE* is defined by the sequence of (<field name>, data type) pairs indicated by the sequence of column descriptors of *NJQE* taken in order.

27) The row type of a <query term> is that of the <non-join query term> or <joined table> that it immediately contains.

28) The row type of a <non-join query term> *NJQT* is determined as follows:

Case:

a) If *NJQT* immediately contains <non-join query primary> *NJQP*, then the row type of *NJQT* is that of *NJQP*.

b) If *NJQT* immediately contains <recursive union> *RU*, then the row type of *NJQT* is that of *RU*.

c) Otherwise:

Case:

i) If no <corresponding spec> is specified, and the <query term> and <query primary> immediately contained in *NJQT* have the same row type, then the row type of *NJQT* is that row type.

ii) Otherwise, the row type of *NJQT* is defined by the sequence of (<field name>, data type) pairs indicated by the sequence of column descriptors of *NJQT* taken in order.

29) The row type of a <query primary> is that of the <non-join query primary> or <joined table> that it immediately contains.

30) The row type of a <non-join query primary> *NJQP* is that of the <simple table> or <non-join query expression> that it immediately contains.

31) The row type of a <simple table> is that of the <query specification>, <table value constructor>, <explicit table>, or <collection expression> that it immediately contains.

32) The row type of an <explicit table> is that of the table identified by the <table name> that it immediately contains.

33) The row type of a <collection expression> is that of the <value expression> that it immediately contains.

34) If the data type of any column of a <query term> is large object string, then ALL shall be specified.

35) If the data type of any column of a <query primary> is large object string, then ALL shall be specified.

Access Rules

None.

General Rules

ISO Only–SQL3

1) If <with clause> is specified, then:

   a) For every <with list element> *WLE*, let *WQN* be the <query name> immediately contained in *WLE*. Let *WQE* be the <query expression> immediately contained in *WLE*. Let *WLT* be the table resulting from evaluation of *WQE*, with each column name replaced by the corresponding element of the <with column list>, if any, immediately contained in *WLE*.

   b) Every <table reference> contained in <query expression> that specifies *WQN* shall identify *WLT*.

2) If a <simple table> is a <collection expression> that has a null value, then the result of <simple table> is an empty table.

ANSI Only—caused by ISO changes not yet considered by ANSI

3) For each <trigger definition> specified, the General Rules of <trigger definition> are effectively applied to the <derived table> defined by the <query expression>.

4) Case:

   a) If no set operator is specified, then *T* is the result of the specified <simple table> or <joined table>.

b) If a set operator is specified, then the result of applying the set operator is a table containing the following rows:

i) Let *R* be a row that is a duplicate of some row in *T1* or of some row in *T2* or both. Let *m* be the number of duplicates of *R* in *T1* and let *n* be the number of duplicates of *R* in *T2*, where $m \geq 0$ and $n \geq 0$.

ANSI Only—caused by ISO changes not yet considered by ANSI

ii) If both *T1* and *T2* are list tables, then *T* is a list table.

iii) If ALL is not specified, then

Case:

1) If UNION is specified, then

Case:

A) If $m > 0$ or $n > 0$, then *T* contains exactly one duplicate of *R*.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a list table, then *R* is derived from its first duplicate occurring in *T1*, or, failing that, from its first duplicate occurring in *T2*.

B) Otherwise, *T* contains no duplicate of *R*.

2) If EXCEPT is specified, then

Case:

A) If $m > 0$ and $n = 0$, then *T* contains exactly one duplicate of *R*.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a list table, then *R* is derived from its first duplicate occurring in *T1*.

B) Otherwise, *T* contains no duplicate of *R*.

3) If INTERSECT is specified, then

Case:

A) If $m > 0$ and $n > 0$, then *T* contains exactly one duplicate of *R*.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a list table, then *R* is derived from its first duplicate occurring in *T1*.

B) Otherwise, *T* contains no duplicates of *R*.

iv) If ALL is specified, then

Case:

1) If UNION is specified, then the number of duplicates of *R* that *T* contains is ($m + n$).

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a list table, then these duplicates of *R* are derived from all its duplicates occurring in *T1* and *T2*.

2) If EXCEPT is specified, then the number of duplicates of *R* that *T* contains is the maximum of ($m - n$) and 0.

ANSI Only—caused by ISO changes not yet considered by ANSI

If *T* is a list table and contains any duplicates of *R*, they they are derived from the first $m - n$ of its duplicates occurring in *T1*.

3) If INTERSECT is specified, then the number of duplicates of *R* that *T* contains is the minimum of *m* and *n*.

ANSI Only—caused by ISO changes not yet considered by ANSI

Let *d* be the minimum of *m* and *n*. If *T* is a list table and contains any duplicates of *R*, then they are derived from the first *d* of its duplicates occurring in *T1*.

ANSI Only—caused by ISO changes not yet considered by ANSI

v) If *T* is a list table, then its rows are in the order of those of *T1* from which they are derived, followed by the order of those of *T2* from which they are derived.

Note: See the General Rules of Subclause 8.2, "<comparison predicate>".

5) If a set operator is specified, then for each column whose data type is interval, let *UDT* be in turn the data type of the corresponding column of *T* and let *SV* be the value of the column in each row of the first and second operands. The value of the corresponding column of *T* in the corresponding row of *T* is

CAST (*SV* AS *UDT*)

6) Case:

   a) If the set operator UNION is specified, then:

      i) If a row *R* is deleted from *T*, then

         Case:

         1) If *R* is derived from *T1*, then the row from which R is derived is deleted from *T1*.

         2) Otherwise, the row from which *R* is derived is deleted from *T2*.

      ii) If a row *R* is replaced by some row *RR*, then

         Case:

         1) If *R* is derived from *T1*, then the row of *T1* from which *R* is derived is replaced by *RR*.

         2) Otherwise, the row of *T2* from which *R* is derived is replaced by *RR*.

   b) If EXCEPT is specified and a row *R* of *T* is replaced by some row *RR*, then the row of *T1* from which *R* is derived is replaced by *RR*.

   c) If INTERSECT is specified, then:

      i) If a row *R* is inserted into *T*, then:

         1) If *T1* does not contain a row whose value equals the value of *R*, then *R* is inserted into *T1*.

         2) If *T1* contains a row whose value equals the value of *R* and no row of *T* is derived from that row, then *R* is inserted into *T1*.

         3) If *T2* does not contain a row whose value equals the value of *R*, then *R* is inserted into *T2*.

         4) If *T2* contains a row whose value equals the value of *R* and no row of *T* is derived from that row, then *R* is inserted into *T2*.

      ii) If a row *R* is replaced by some row *RR*, then:

         1) The row of *T1* from which *R* is derived is replaced with *RR*.

         2) The row of *T2* from which *R* is derived is replaced with *RR*.

ISO Only—caused by ANSI changes not yet considered by ISO

7) *T* is effectively a base table with respect to updatability rules.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <non-join query term> shall contain no <recursive union>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <simple table> shall not be a <table value constructor> except in an <insert statement>.

   b) Conforming Intermediate SQL shall contain no <explicit table>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <query expression> shall not specify EXCEPT.

   b) A <query term> shall not specify INTERSECT.

   c) A <query expression> shall not contain a <joined table>.

   d) A <query expression> shall not specify CORRESPONDING.

   e) If UNION is specified, then except for column names, the descriptors of the first and second operands shall be identical and the descriptor of the result is identical to the descriptor of the operands.

7.15 <recursive union>

Function

Specify an expression to invoke a limited amount of recursion.

Format

```
<recursive union> ::=
       <left paren> <initial expression>
         RECURSIVE UNION <correlation name list>
             [ <left paren> <recursive column list> <right paren> ]
         <iteration expression>
         [ <search clause> ]
         [ <cycle clause> ]
         [ <limit clause> ] <right paren>

<initial expression> ::= <query expression>

<iteration expression> ::= <query expression>

<correlation name list> ::=
       <correlation name> [ { <comma> <correlation name> }... ]

<recursive column list> ::= <column name list>

<search clause> ::=
       SEARCH <search order> SET <sequence column>

<search order> ::=
         PREORDER
       | { DEPTH | BREADTH } FIRST BY <sort specification list>

<sequence column> ::= <column name>

<cycle clause> ::=
       CYCLE [ <cycle column list> ]
         SET <cycle mark column> [ TO <cycle mark value> ]

<cycle column list> ::=
       <cycle column> [ { <comma> <cycle column> }... ]

<cycle column> ::= <column name>

<cycle mark column> ::= <column name>

<cycle mark value> ::= <value expression>

<limit clause> ::=
       [ RETURN | EXCEPTION ] LIMIT <left paren> <value specification> <right paren>
```

Syntax Rules

1) The scope of each <correlation name> in the <correlation name list> is the <iteration expression>.

2) No <correlation name> shall be specified more than once in the <correlation name list>.

3) The degree of the <query expression> in <initial expression> shall be the same as the degree of the <query expression> in <iteration expression>.

4) The data type of the *i*-th column of the <query expression> in <initial expression> and the data type of the *i*-th column of the <query expression> in <iteration expression> shall be comparable.

5) If a <recursive column list> is specified, then the same <column name> shall not be specified more than once in the <recursive column list>, and the number of <column name>s in the <recursive column list> shall be the same as the degree of the table specified by the <initial expression>.

6) The result description of the <recursive union> includes the column descriptions of the unions of the <initial expression> and the <iteration expression>. If a <recursive column list> is specified, then the name of the *i*-th column of the result is the *i*-th <column name> in the <recursive column list>. If a <recursive column list> is not specified, then the name of the *i*-th column of the result is the name of the *i*-th column of the table specified by the <initial expression>. A *result row* is a row having the result description.

7) The row type of the result is determined as follows:

   Case:

   a) If no <recursive column list> is specified, and the <initial expression> and <iteration expression> immediately contained in <recursive union> have the same row type, then the row type of the result is that row type.

   b) Otherwise the row type of the result is defined by the sequence of (<field name>, data type) pairs of the columns of the result of the <recursive union> taken in order.

8) If a <search clause> is specified, then:

   a) The <sequence column> shall identify a column of the result description whose data type is INTEGER.

   b) A column reference specified in a <sort specification> shall not contain an <item qualifier> and shall identify a column of the result description.

   c) An <unsigned integer> specified as a <sort specification> shall not be greater than the degree of the result description.

Note: column reference is the preferred designator for a column in a <sort specification>. The designator <unsigned integer> is a deprecated feature that is supported for compatibility with earlier versions of the
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

9) If a <cycle clause> is specified, then:

   a) Case:

      i) If a <cycle column list> is specified, then each <column name> in that <cycle column list> shall identify a column of the combination description.

      ii) If a <cycle column list> is not specified, then let *referenced <column name>s* be the <column name>s of the combination description that are referenced by outer references in the <iteration expression>. There shall be at least one referenced <column name>. A <cycle column list> containing exactly one instance of each referenced <column name> is implicit.

      Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

   b) The <cycle mark column> shall identify a column of the result whose data type is character string of length 1.

   c) If a <cycle mark value> is specified, then its data type shall be character string of length 1. If a <cycle mark value> is not specified, then '1' is implicit.

10) Case:

   a) If a <limit clause> is specified, then:

      i) The data type of the <value specification> shall be exact numeric with scale 0.

      ii) If neither RETURN nor EXCEPTION is specified, then EXCEPTION is implicit.

   b) If a <limit clause> is not specified, then EXCEPTION LIMIT (*k*) is implicit, where *k* is an implementation-defined positive integer.

11) Let *N* be the number of <correlation name>s in the <correlation name list>. The combination description of the <recursive union> is a table description that is the concatenation of *N* result descriptions. The *i*-th <correlation name> in the <correlation name list> designates the *i*-th instance of the result description in the combination description. A *combination row* is a row having the combination description.

12) A *linear recursive union* is a <recursive union> whose <correlation name list> contains exactly one <correlation name>. A *non-linear recursive union* is a <recursive union> whose <correlation name list> contains two or more <correlation name>s.

Access Rules

None.

General Rules

1) If a linear recursive union is specified, then:

   a) Let *IT* be the set of rows that is the result of evaluating the <initial expression>.

   b) If *IT* is the empty set, then that empty set is the result of the <recursive union> and the remaining General Rules are not applicable.

   c) Let *L* be the result of the LIMIT <value specification>.

d) If *L* is not positive, then an exception condition is raised: *data exception—invalid limit value*.

e) Let *SEQUENCE* be an INTEGER initialized to 0.

f) Let *RT* be an empty set to contain result rows. Each member of RT will be a result row.

g) Let *S* be an empty push-down stack. Each entry on *S* will be a result row.

h) While TRUE:

i) Depending on whether DEPTH, BREADTH, or PREORDER is specified, the *specified search order* is the order specified by the <sort specification list>, the reverse of the order specified by the <sort specification list>, or an implementation-defined order, respectively.

ii) Push each row of *IT* onto *S*, in the specified search order.

iii) If *S* is empty, then exit the While loop.

iv) If *SEQUENCE* = *L*, then

Case:

1) If RETURN is specified, then exit the While loop.

2) if EXCEPTION is specified, then an exception condition is raised: *cardinality violation*.

v) Increment *SEQUENCE* by 1.

vi) A row *R1* is the *immediate predecessor* of each row in a set of rows that is the result of evaluating the <iteration expression> with the <correlation name> bound to that row *R1*. A row *R2* is a *predecessor* of a row *R3* if *R1* is the immediate predecessor of *R3* or if *R1* is the immediate predecessor of a row *R2* that is a predecessor of *R3*.

vii) Depending on whether DEPTH, BREADTH, or PREORDER is specified, pop the last, the first, or an arbitrary row *R* of *S*, respectively.

viii) If a <sequence column> is specified, then set the column of *R* identified by the <sequence column> to the value of *SEQUENCE*.

ix) Case:

1) If a <cycle clause> is specified and there is a predecessor of *R* for which each column designated by a <cycle column> is equal to the corresponding column of *R*, then:

A) Set the column of *R* designated by the <cycle mark column> to the value of the <cycle mark value>.

B) Let *IT* be an empty set.

2) Otherwise, let *IT* be the set of rows that is the result of evaluating the <iteration expression> with the <correlation name> bound to *R*.

x) Insert *R* into *RT*.

End While.

i) *RT* is the result of the <recursive union>.

Note: For a linear recursive union, the preceding General Rule and the following General Rule have the same effect. The special Rules for the linear case are, however, simpler than the General Rules that cover both the linear and non-linear cases, and so the special Rules for the linear case are included, even though they are redundant.

2) If a non-linear recursive union is specified, then:

a) Let *IT* be the set of rows that is the result of evaluating the <initial expression>.

b) If *IT* is the empty set, then that empty set is the result of the <recursive union> and the remaining General Rules are not applicable.

c) Let *L* be the result of the LIMIT <value specification>.

d) If *L* is not positive, then an exception condition is raised: *data exception—invalid limit value*.

e) Let *SEQUENCE* be an INTEGER initialized to 0.

f) Let *RT* be an empty set to contain result rows. Each member of RT will be a result row.

g) Let *S* be an empty push-down stack. Each entry on *S* will be either a result row or a combination row.

h) Let *WL* be an empty ordered list. Each entry on *WL* will contain a result row and an *OUTPUT-STATUS* that is either *output* or *not* output . For a result row *RR*, the *entry for RR in WL* refers to the entry of *WL* whose result row is equal to *RR*.

i) Let *R* be an empty combination row.

j) While TRUE:

i) For result rows, the *natural search order* is the order specified by the <sort specification list>. For combination rows, the (natural search order) is the order specified by repeating the <sort specification list> *N* times, with the *i*-th <correlation name> as an <item qualifier> of each <column name> in the *i*-th repetition. Depending on whether DEPTH, BREADTH, or PREORDER is specified, the (specified search order) is the natural search order, the reverse of the natural search order, or an implementation-defined search order, respectively.

ii) For every row *ITR* of *IT*, in the specified order:

1) Push *ITR* onto *S* as a result row.

2) Append *ITR* to *WL* with an *OUTPUT-STATUS* of *not* output .

iii) If *S* is empty, then exit the While loop.

iv) Depending on whether DEPTH, BREADTH, or PREORDER is specified, pop the last, the first, or an arbitrary row *R* of *S*, respectively.

v) Case:

1) If *R* is a result row, then perform NODE(*R*).

2) If *R* is a combination row, then perform COMBINATION(*R*).

End While.

k) *RT* is the result of the <recursive union>.

l) PROCEDURE NODE (*R*):

Argument *R* is a result row.

The NODE procedure references *WL* and *S* as global variables.

i) At the beginning of each iteration of the While of the General Rules of Subclause 7.15, "<recursive union>", each result constituent of combination row *R* is an *immediate constituent* of each result row of *IT*. A result row *R1* is a *predecessor* of a result row *R3* if *R1* is an immediate predecessor of *R3* or if *R1* is an immediate predecessor of a row *R2* that is a predecessor of *R3*.

ii) A combination row *C1* is a *cycle match* of a combination row *C2* if for each <cycle column> *C*, the value designated by *C* in *C1* is equal to the value designated by *C* in *C2*.

iii) If a <cycle clause> is specified and there is a predecessor of *R* that is a cycle match of *R*, then:

1) Set the column of *R* designated by the <cycle mark column> to the <cycle mark value>.

2) Perform OUTPUT(*R*).

3) Return from the NODE procedure.

iv) Perform OUTPUT(*R*).

v) Let *PT* be the set of all result rows prior to the entry for *R* in *WL*.

vi) Let a *C-term* be an extended Cartesian product with *N* operands, each operand being either *R* or *PT* and at least one operand being *R*. There are $2^N - 1$ different ways to form a *C-term*. Let *CT* be the UNION ALL of the $2^N - 1$ different *C-terms*, with the combination description.

Note: *CT* and the *C-terms* have the following form:

```
           R  x...x R  x R  x R
UNION ALL R  x...x R  x R  x PT
UNION ALL R  x...x R  x PT x R
UNION ALL R  x...x R  x PT x PT
UNION ALL R  x...x PT x R  x R
UNION ALL R  x...x PT x R  x PT
UNION ALL R  x...x PT x PT x R
UNION ALL R  x...x PT x PT x PT
UNION ALL ...
UNION ALL PT x...x PT x PT x R
```

Note: On the first iteration of the While, *PT* is empty and so *CT* is simply the concatenation of *R* with itself *N* times.

Note: If *N* is 1, then there is only one *C-term*, which is *R*, so *CT* is simply *R*.

vii) Each combination row *CR* of *CT* is composed of a sequence of *N* result rows. Each such result row is called a *result constituent* of *CR*.

viii) Push each row of *CT* onto *S* as a combination row, in the specified search order.

ix) Return from the NODE procedure.

m) PROCEDURE COMBINATION (*R*):

Argument *R* is a combination row.

The COMBINATION procedure references *WL* and *SEQUENCE* as global variables.

i) If a <sequence column> is specified, then:

1) Set the <sequence column> of each result constituent of *R* whose entry *AR* on *WL* has *OUTPUT-STATUS* of *output* to the value of the <sequence column> of *AR*.

2) Set the <sequence column> of the *i*-th result constituent of *R* whose entry on *WL* has *OUTPUT-STATUS* of *not* output to *SEQUENCE*+*i*.

ii) Let *IT* be the set of rows that is the result of evaluating the <iteration clause> with the <correlation name>s bound to the result constituents of *R*.

iii) Case:

1) If *IT* is not empty, then for each result constituent *RCC* of *R* whose entry on *WL* has *OUTPUT-STATUS* of *not* output , perform OUTPUT(*RCC*).

2) If *IT* is empty, then perform no action.

3) Return from the COMBINATION procedure.

n) PROCEDURE OUTPUT (*R*):

Argument *R* is a result row.

The OUTPUT procedure references *RT*, *WL*, and *SEQUENCE* as global variables.

i) If the *OUTPUT-STATUS* of *R* on *WL* is *output* , then perform no action and return from the OUTPUT procedure.

ii) If *SEQUENCE* = *L*, then

Case:

1) If RETURN is specified, then exit the <recursive expression> with *RT* as the result.

2) If EXCEPTION is specified, then an exception condition is raised: *cardinality violation*.

iii) Increment *SEQUENCE* by 1.

iv) If a <sequence column> is specified, then set the <sequence column>s of *R* and of the entry for *R* on *WL* to *SEQUENCE*.

v) Insert *R* into *RT*.

vi) Set the *OUTPUT-STATUS* of the entry for *R* on *WL* to *output* .

vii) Return from the OUTPUT procedure.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <recursive union>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.16 <scalar subquery>, <row subquery>, and <table subquery>

Function

Specify a scalar value, a row, or a table derived from a <query expression>.

Format

```
<scalar subquery> ::= <subquery>

<row subquery> ::= <subquery>

<table subquery> ::= <subquery>

<subquery> ::= <left paren> <query expression> <right paren>
```

Syntax Rules

1) The degree of a <scalar subquery> shall be 1.

2) The degree of a <row subquery> shall be greater than 1.

3) Let *QE* be the <query expression> immediately contained in <subquery>.

4) The data type and null class of a <scalar subquery> are the data type and null class of the column of *QE*.

5) The data types and null classes of the columns of a <row subquery> or <table subquery> are the data types and null classes of the respective columns of *QE*.

6) The data type of a <row subquery> is the row type of *QE*.

7) The
   ANSI data type
   ISO row type
   of a <table subquery> is the
   ANSI data type
   ISO row type
   of *QE*.

Access Rules

None.

General Rules

1) If the cardinality of a <scalar subquery> or a <row subquery> is greater than 1, then an exception condition is raised: *cardinality violation*.

2) During the evaluation of a <subquery>, an *atomic execution context* is active. When the <subquery> completes, all savepoints that have been established during its evaluation are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If a <subquery> is contained in a <comparison predicate>, then the <table expression> in the <query specification> shall not contain a <group by clause> or a <having clause> and shall not identify a grouped view.

   b) The <query expression> contained in a <subquery> shall be a <query specification>.

   c) If a <table subquery> is simply contained in an <exists predicate>, then the <select list> of the <query specification> directly contained in the <table subquery> shall comprise either an <asterisk> or a single <derived column>.

8 Predicates

8.1 <predicate>

Function

Specify a condition that can be evaluated to give a boolean value.

Format

```
<predicate> ::=
        <comparison predicate>
      | <between predicate>
      | <in predicate>
      | <like predicate>
      | <null predicate>
      | <quantified comparison predicate>
      | <exists predicate>
      | <unique predicate>
      | <match predicate>
      | <overlaps predicate>
      | <similar predicate>
      | <quantified predicate>
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
      | <there is predicate>
```

```
      | <distinct predicate>
      | <boolean predicate>
```

ANSI Only--SQL3

```
      | <type predicate>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) The result of a <predicate> is a truth value derived according to the General Rules of Subclause 8.2, "<comparison predicate>", Subclause 8.3, "<between predicate>", Subclause 8.4, "<in predicate>", Subclause 8.5, "<like predicate>", Subclause 8.7, "<null predicate>", Subclause 8.8, "<quantified comparison predicate>", Subclause 8.9, "<exists predicate>", Subclause 8.10, "<unique predicate>", Subclause 8.11, "<match predicate>", Subclause 8.6, "<similar predicate>", Subclause 8.13, "<quantified predicate>",
ANSI Subclause 8.14, "<there is predicate>",
or Subclause 8.12, "<overlaps predicate>", as appropriate.

****Editor's Note****

Amelia Carlson has noticed that <quantified comparison predicate> can be reached both directly via <predicate> (see the Format of this Subclause) and indirectly from <quantified predicate> (see the Format of Subclause 8.8, "<quantified comparison predicate>"). It should probably be eliminated from the Format of this Subclause or from the Format of Subclause 8.8, "<quantified comparison predicate>", with appropriate changes to the General Rules and Leveling Rules. See Possible Problem **262**.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <similar predicate>.

   b) Conforming Full SQL language shall contain no <quantified predicate>.

ANSI Only—caused by ISO changes not yet considered by ANSI

   c) Conforming Full SQL language shall contain no <there is predicate>.

   d) Conforming Full SQL language shall contain no <function invocation predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <predicate> shall not be a <match predicate>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

   b) Conforming Entry SQL language shall not contain any <unique predicate>.

   c) Conforming Entry SQL language shall not contain any <distinct predicate>.

8.2 <comparison predicate>

Function

Specify a comparison of two row values.

Format

```
<comparison predicate> ::=
       <row value expression> <comp op> <row value expression>

<comp op> ::=
         <equals operator>
       | <not equals operator>
       | <less than operator>
       | <greater than operator>
       | <less than or equals operator>
       | <greater than or equals operator>
```

Syntax Rules

1) if the data type of a pair of corresponding values is a large object string, then <comp op> shall be either <equals operator> or <not equals operator>.

2) The two <row value expression>s shall be of the same degree.

3) Let R_x and R_y be the two <row value expression>s.

   Case:

   a) If the <comp op> is <not equals operator>, then the <comparison predicate> is equivalent to:

      NOT(R_x = R_y)

   b) If the <comp op> is <greater than operator>, then the <comparison predicate> is equivalent to:

      (R_y < R_x)

   c) If the <comp op> is <less than or equals operator>, then the <comparison predicate> is equivalent to:

      (R_x < R_y
      OR
      R_y = R_x)

   d) If the <comp op> is <greater than or equals operator>, then the <comparison predicate> is equivalent to:

      (R_y < R_x
      OR
      R_y = R_x)

4) Let *corresponding values* be values with the same ordinal position in the two <row value expression>s.

5) The data types of the corresponding values of the two <row value expression>s shall be comparable.

6) If any pair of respective values has a <collection type>, then <comp op> shall be either <equals operator> or <not equals operator>.

7) If any pair of respective values has a data type that is an abstract data type, and <less than operator>, <greater than operator>, <less than or equals operator>, or <greater than or equals operator> is specified, then the abstract data type shall not have been defined with an <abstract data type definition> that specifies an <ordering clause> that specifies an <equals function specification> and does not specify a <less-than function specification>.

8) Let *X* be a value in the first <row value expression> and *Y* be the corresponding value in the second <row value expression>. If *X* and *Y* have data type character string, then the pair-wise comparison collating sequence used to compare *X* and *Y* is determined by the table for collating sequences for comparisons (Subclause 4.2.3, "Rules determining collating sequence usage"). For any pair of corresponding character strings, let *CS* be the identified collating sequence.

9) If all the values simply contained in the two <row value expression>s that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

ANSI Only–SQL3

10) If any pair of respective values in the <row value expression>s in a <comparison predicate> has a row identifier data type, then the comparison operator shall not be <less than operator>, <greater than operator>, <less than or equals operator>, or <greater than or equals operator>.

Access Rules

None.

General Rules

1) Let *XV* and *YV* be any two corresponding values of the two row values.

   Case:

   a) If the data type of *X* is a distinct type, then let *XDV* and *YDV* be the values reprented by *X* and *Y*, respectively, and let *XV* and *YV* be the representations of *XDV* and *YDV*, respectively, in the data type of the source type of *X* and the source type of *Y*, respectively.

      Note: *Source type* is defined in Subclause 11.49, "<distinct type definition>".

   b) Otherwise, let *XV* and *YV* be the values represented by *X* and *Y*, respectively.

2) Case:

   a) If the data type of *X* is an abstract data type, then then let *ADT* be the data type that is the most specific unique type that is a supertype of the data types of *X* and *Y*. Equality

is determined by invoking the function identified in the <ordering clause> of the abstract data type descriptor that describes *ADT*, and greater-than and less-than relationships are determined by invoking one of relative ordering function, hash ordering function, or less-than ordering function of the abstract data type descriptor that describes *ADT*.

Note: If both "*X* = *Y*" and "*X* < *Y*" are false, then "*X* > *Y*" is true.

Note: *Most specific unique type* is defined in Subclause 4.11.5, "Subtypes and supertypes for ADTs".

b) If *XV* or *YV* is a null value, then

Case:

i) If the result has the general null class, then the result of "*X* <comp op> *Y*" is the general null value.

ii) Otherwise, the result of "*X* <comp op> *Y*" is the minimum of the null values of the values simply contained in the two <row value expression>s.

c) If *XV* and *YV* are non-null values, then "*X* <comp op> *Y*" is true or false as follows:

i) "*X* = *Y*" is true if and only if *XV* and *YV* are equal.

- 1 subrule deleted.

ii) "*X* < *Y*" is true if and only if *XV* is less than *YV*.

- 3 subrules deleted.

iii) "*X* <comp op> *Y*" is false if and only if "*X* <comp op> *Y*" is not true.

3) Numbers are compared with respect to their algebraic value.

4) Enumerated type values are compared with respect to the ordering specified by the <enumerated type>.

5) The comparison of two character strings is determined as follows:

a) If the length in characters of *X* is not equal to the length in characters of *Y*, then the shorter string is effectively replaced, for the purposes of comparison, with a copy of itself that has been extended to the length of the longer string by concatenation on the right of one or more pad characters, where the pad character is chosen based on *CS*. If *CS* has the NO PAD attribute, then the pad character is an implementation-dependent character different from any character in the character set of *X* and *Y* that collates less than any string under *CS*. Otherwise, the pad character is a <space>.

b) The result of the comparison of *X* and *Y* is given by the collating sequence *CS*.

c) Depending on the collating sequence, two strings may compare as equal even if they are of different lengths or contain different sequences of characters. When any of the operations MAX, MIN, and DISTINCT reference a grouping column, and the UNION, EXCEPT, and INTERSECT operators refer to character strings, the specific value selected by these operations from a set of such equal values is implementation-dependent.

Note: If the coercibility attribute of the comparison is *Coercible*, then the collating sequence used is the default defined for the character repertoire. See also other Syntax Rules in this Subclause, Subclause 10.5, "<character set specification>", and Subclause 11.37, "<character set definition>".

6) The comparison of two binary string values, *X* and *Y*, is determined by comparison of their octets with the same ordinal position. If X_i and Y_i are the values of the i-th octets of *X* and *Y*, respectively, and if L_x is the length in octets of *X* AND L_y is the length in octets of *Y*, then *X* is equal to *Y* if and only if $L_y = L_y$ and if $X_i = Y_i$ for all *i*.

7) The comparison of two bit string values, *X* and *Y*, is determined by comparison of their bits with the same ordinal position. If X_i and Y_i are the values of the *i*-th bits of *X* and *Y*, respectively, and if L_X is the length in bits of *X* and L_Y is the length in bits of *Y*, then:

   a) *X* is equal to *Y* if and only if $L_X = L_Y$ and $X_i = Y_i$ for all *i*.

   b) *X* is less than *Y* if and only if:

      i) $L_X < L_Y$ and $X_i = Y_i$ for all *i* less than or equal to L_X; or

      ii) $X_i = Y_i$ for all $i < n$ and $X_n = 0$ and $Y_n = 1$ for some *n* less than or equal to the minimum of L_X and L_Y.

8) The comparison of two datetimes is determined according to the interval resulting from their subtraction. Let *X* and *Y* be the two values to be compared and let *H* be the least significant <datetime field> of *X* and *Y*. The result of *X* <comp op> *Y* is defined as:

   (*X* − *Y*) *H* <comp op> INTERVAL (0) *H*

   Note: Two datetimes are comparable only if they have the same <datetime field>s; see Subclause 4.9.1, "Datetimes".

9) The comparison of two intervals is determined by the comparison of their corresponding values after conversion to integers in some common base unit. Let *X* and *Y* be the two intervals to be compared. Let *A* TO *B* be the specified or implied datetime qualifier of *X* and *C* TO *D* be the specified or implied datetime qualifier of *Y*. Let *T* be the least significant <datetime field> of *B* and *D* and let *U* be a datetime qualifier of the form *T*(*N*), where *N* is an <interval leading field precision> large enough so that significance is not lost in the CAST operation.

   X is effectively replaced by CAST (*X* AS INTERVAL *U*).

   Y is effectively replaced by CAST (*Y* AS INTERVAL *U*).

   The result of the comparison is effectively computed as:

   CAST (*X* AS INTEGER) <comp op> CAST (*Y* AS INTEGER)

10) In comparisons of boolean values, *true* is greater than *false*

11) The comparison of two sets or multisets is effectively computed as follows:

   a) Let *S1* be the first set or multiset; let *S2* be the second set or multiset.

   b) Let *TEMPS1* be a copy of *S1*; let *TEMPS2* be a copy of *S2*.

   c) For each element of *TEMPS1*, if that element is equal to an element in *TEMPS2*, then delete that element from *TEMPS1* and *TEMPS2*.

   d) Case:

      i) If *TEMPS1* is empty and *TEMPS2* is empty, then "*S1* = *S2*" is true.

ii) If the number of elements in *TEMPS1* equals the number of elements in *TEMPS2* and the number of non-null elements in *TEMPS1* is less than or equal to the number of null elements in *TEMPS2* and the number of non-null elements in *TEMPS2* is less than or equal to the number of null elements in *TEMPS1*, then "*S1* = *S2*" is unknown.

iii) Otherwise, "*S1* = *S2*" is false.

12) The comparison of two lists is effectively computed as follows:

a) Let *L1* be the first list; let *L2* be the second list.

b) Let *N1* be the number of elements in *L1*; let *N2* be the number of elements in *L2*.

c) Let $E1_i$ be the *i*-th element of *L1*; let $E2_i$ be the *i*-th element of *L2*.

d) If *N1* is not equal to *N2*, then "*L1* = *L2*" is false.

e) If *N1* equals zero and *N2* equals zero, then "*L1* = *L2*" is true.

f) If "$E1_i = E2_i$" is true for all *i* from $i = 1$ to $i = N1$, then "*L1* = *L2*" is true.

g) If "$E1_i = E2_i$" is false for some *i* from *i=1* to *i=N1*, then "*L1* = *L2*" is false.

h) Otherwise, "*L1* = *L2*" is unknown.

13) Let R_x and R_y be the two <row value constructor>s of the <comparison predicate> and let RX_i and RY_i be the *i*-th values of R_x and R_y, respectively. "R_x <comp op> R_y" is true, false, or unknown as follows:

a) "$R_x = R_y$" is true if and only if $RX_i = RY_i$ for all *i*.

b) "$R_x < R_y$" is true if and only if $RX_i = RY_i$ for all $i < n$ and $RX_n < RY_n$ for some *n*.

c) "$R_x = R_y$" is false if and only if "NOT$R_x = R_y$" for some *i*.

d) "$R_x < R_y$" is false if and only if "$R_x = R_y$" or "$R_x < R_y$".

- 3 subrules deleted.

e) "R_x <comp op> R_y" is unknown if and only if "R_x <comp op> R_y" is neither true nor false.

ANSI Only–SQL3

14) If the data type of the two operands of the <equals operator> are <row identifier type>s, then *X=Y* if and only if they identify the same row of the same base table.

Note: Casting a row identifer value to the row identifier data type of another table will not affect its value or consequently the equality test.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.3 <between predicate>

Function

Specify a range comparison.

Format

```
<between predicate> ::=
      <row value expression> [ NOT ] BETWEEN
        <row value expression> AND <row value expression>
```

Syntax Rules

1) The data type of every respective value shall not be large object string.

2) The three <row value expression>s shall be of the same degree.

3) In the three <row value expression>s, the data types of values with the same ordinal position shall be the same.

4) Let *X*, *Y*, and *Z* be the first, second, and third <row value expression>s, respectively.

5) "*X* NOT BETWEEN *Y* AND *Z*" is equivalent to "NOT (*X* BETWEEN *Y* AND *Z*)".

6) "*X* BETWEEN *Y* AND *Z*" is equivalent to "*X*>=*Y* AND *X*<=*Z*".

7) If all the values simply contained in the two <row value expression>s that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.4 <in predicate>

Function

Specify a quantified comparison.

Format

```
<in predicate> ::=
       <row value expression>
         [ NOT ] IN <in predicate value>

<in predicate value> ::=
         <table subquery>
       | <left paren> <in value list> <right paren>

<in value list> ::=
       <value expression> { <comma> <value expression> }...
```

Syntax Rules

1) The data type of each <value expression> in the <row value constructor>, the <in value list>, and the <table subquery> shall not be large object string.

2) Let *IVL* be an <in value list>.

   (*IVL*)

   is equivalent to the <table value constructor>:

   (VALUES *IVL*)

3) Let *RVC* be the <row value expression> and let *IPV* be the <in predicate value>.

4) The expression

   RVC NOT IN *IPV*

   is equivalent to

   NOT (*RVC* IN *IPV*)

5) The expression

   RVC IN *IPV*

   is equivalent to

   RVC = ANY *IPV*

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain a <value expression> in an <in value list> that is not a <value specification>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.5 <like predicate>

Function

Specify a pattern-match comparison.

Format

```
<like predicate> ::=
         <character like predicate>
       | <octet like predicate>

<character like predicate> ::=
       <character match value> [ NOT ] LIKE <character pattern>
         [ ESCAPE <escape character> ]

<character match value> ::= <character value expression>

<character pattern> ::= <character value expression>

<escape character> ::= <character value expression>

<octet like predicate> ::=
       <octet match value> [ NOT ] LIKE <octet pattern>
         [ ESCAPE <escape octet> ]

<octet match value> ::= <blob value expression>

<octet pattern> ::= <blob value expression>

<escape octet> ::= <blob value expression>
```

Syntax Rules

1) The data types of <character match value>, <character pattern>, and <escape character> shall be character string. <character match value>, <character pattern>, and <escape character> shall be comparable.

2) The data types of <octet match value>, <octet pattern>, and <escape octet> shall be binary string.

3) If <character like predicate> is specified, then:

   a) Let *MC* be the result of the <character value expression> of the <character match value>, let *PC* be the result of the <character value expression> of the <character pattern>, and let *EC* be the result of the <character value expression> of the <escape character> if one is specified.

   b) "*MC* NOT LIKE *PC*" is equivalent to "NOT (*MC* LIKE *PC*)".

   c) Case:

      i) If <escape character> is not specified, then the collating sequence used for the <like predicate> is determined by Table 3, "Collating sequence usage for comparisons", taking <character match value> as comparand 1 and <character pattern> as comparand 2.

ii) Otherwise, let *C1* be the coercibility attribute and collating sequence of the <character match value>, and *C2* be the coercibility attribute and collating sequence of the <character pattern>. Let *C3* be the resulting coercibility attribute and collating sequence as determined by Table 2, "Collating coercibility rules for dyadic operators", taking *C1* as the operand 1 coercibility and *C2* as the operand 2 coercibility. The collating sequence used for the <like predicate> is determined by Table 3, "Collating sequence usage for comparisons", taking *C3* as the coercibility attribute and collating sequence of comparand 1 and <escape character> as comparand 2.

d) If all of the values *MC*, *PC*, and *EC* that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

4) If <octet like predicate> is specified, then:

a) Let *MB* be the result of the <blob value expression> of the <octet match value>, let *PB* be the result of the <blob value expression> of the <octet pattern>, and let *EB* be the result of the <blob value expression> of the <escape octet> if one is specified.

b) "*MB* NOT LIKE *PB*" is equivalent to "NOT (*MB* LIKE *PB*)"

c) If all of the values *MB*, *PB*, *EB* that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rule

1) If <character like predicate> is specified, then:

a) Case:

i) If ESCAPE is not specified and either *MC* or *PC* are null values, then

Case:

1) If the result has the general null class, then the result of

MC LIKE *PC*

is the general null value.

2) Otherwise, the result of

MC LIKE *PC*

is the minimum of the null values of *MC* and *PC*.

ii) If ESCAPE is specified and one or more of *MC*, *PC* and *EC* are null values, then

Case:

1) If the result has the general null class, then the result of

MC LIKE *PC* ESCAPE *EC*

is the general null value.

2) Otherwise the result of

MC LIKE *PC* ESCAPE *EC*

is the minimum of the null values of *MC*, *PC*, and *EC*.

Note: If none of *MC*, *PC*, and *EC* (if present) are null values, then the result is either true or false.

b) Case:

i) If an <escape character> is specified, then:

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

2) If there is not a partitioning of the string *PC* into substrings such that each substring has length 1 or 2, no substring of length 1 is the escape character *EC*, and each substring of length 2 is the escape character *EC* followed by either the escape character *EC*, an <underscore> character, or the <percent> character, then an exception condition is raised: *data exception—invalid escape sequence*.

If there is such a partitioning of *PC*, then in that partitioning, each substring with length 2 represents a single occurrence of the second character of that substring. Each substring with length 1 that is the <underscore> character represents an arbitrary character specifier. Each substring with length 1 that is the <percent> character represents an arbitrary string specifier. Each substring with length 1 that is neither the <underscore> character nor the <percent> character represents the character that it contains.

ii) If an <escape character> is not specified, then each <underscore> character in *PC* represents an arbitrary character specifier, each <percent> character in *PC* represents an arbitrary string specifier, and each character in *PC* that is neither the <underscore> character nor the <percent> character represents itself.

c) The string *PC* is a sequence of the minimum number of substring specifiers such that each <character representation> of *PC* is part of exactly one substring specifier. A substring specifier is an arbitrary character specifier, an arbitrary string specifier, or any sequence of <character representation>s other than an arbitrary character specifier or an arbitrary string specifier.

d) If either *MC* or *PC* is a null value, then the result of "*MC* LIKE *PC*" is the general null value. If *MC* and *PC* are non-null values, then "*MC* LIKE *PC*" is either true or false.

e) Case:

i) If *MC* and *PC* are character strings whose lengths are variable and if the lengths of both *MC* and *PC* are 0, then

MC LIKE *PC*

is true.

ii) The <predicate>

MC LIKE *PC*

is true if there exists a partitioning of *MC* into substrings such that:

1) A substring of *MC* is a sequence of 0 or more contiguous <character representation>s of *MC* and each <character representation> of *MC* is part of exactly one substring.

2) If the *i*-th substring specifier of *PC* is an arbitrary character specifier, the *i*-th substring of *MC* is any single <character representation>.

3) If the *i*-th substring specifier of *PC* is an arbitrary string specifier, then the *i*-th substring of *MC* is any sequence of 0 or more <character representation>s.

4) If the *i*-th substring specifier of *PC* is neither an arbitrary character specifier nor an arbitrary string specifier, then the *i*-th substring of *MC* is equal to that substring specifier according to the collating sequence of the <like predicate>, without the appending of <space> characters to *MC*, and has the same length as that substring specifier.

5) The number of substrings of *MC* is equal to the number of substring specifiers of *PC*.

iii) Otherwise,

MC LIKE *PC*

is false.

2) If <octet like predicate> is specified, then:

a) Case:

i) If ESCAPE is not specified and either *MB* or *PB* are null values, then

Case:

1) If the result has the general null class, then the result of

MB LIKE *PB*

is the general null value.

2) Otherwise, the result of

MB LIKE *PB*

is the minimum of the null values of *MB* and *PB*.

ii) If ESCAPE is specified and one or more of *MB*, *PB* and *EB* are null values, then

Case:

1) If the result has the general null class, then the result of

MB LIKE *PB* ESCAPE *EB*

is the general null value.

2) Otherwise the result of

MB LIKE *PB* ESCAPE *EB*

is the minimum of the null values of *MB*, *PB*, and *EB*.

Note: If none of *MB*, *PB*, and *EB* (if present) are null values, then the result is either true or false.

b) <percent> in the context of an <octet like predicate> has the the same bit pattern as a <percent> in the SQL_TEXT character repertoire.

c) <underscore> in the context of an <octet like predicate> has the same bit pattern as an <underscore> in the SQL_TEXT character repertoire.

d) Case:

   i) If an <escape octet> is specified, then:

      1) If the length in octets of *EB* is not equal to 1, than an exception condition is raised: *data exception—invalid escape locator*.

      2) If there is not a partitioning of the string *PB* into substrings such that each substring has length 1 or 2, no substring of length 1 is the escape octet *EB*, and each substring of length 2 is the escape octet *EB* followed by either the escape octet *EB*, an <underscore> octet, or the <percent> octet, then an exception condition is raised: *data exception—invalid escape sequence*.

      If there is such a partitioning of *PB*, then in that partitioning, each substring with length 2 represents a single occurrence of the second octet of that substring. Each substring with length 1 that is the <underscore> octet represents an arbitrary octet specifier. Each substring with length 1 that is the <percent> octet represents an arbitrary string specifier. Each substring with length 1 that is neither the <underscore> octet nor the <percent> octet represents the octet that it contains.

   ii) If an <escape octet> is not specified, then each <underscore> octet in *PB* represents an arbitrary octet specifier, each <percent> octet in *PB* represents an arbitrary string specifier, and each octet in *PB* that is neither the <underscore> octet nor the <percent> octet represents itself.

e) The string *PB* is a sequence of the minimum number of substring specifiers such that each portion of *PB* is part of exactly one substring specifier. A substring specifier is an arbitrary octet specifier, and arbitrary string specifier, or any sequence of octets other than an arbitrary octet specifier or an arbitrary string specifier.

f) If either *MB* or *PB* is a null value, then the result of "*MB* LIKE *PB*" is the general null value. If *MB* and *PB* are nonnull values, then "*MB* LIKE *PB*" is either true or false.

g) Case:

   i) If the lengths of both *MB* and *PB* are 0, then

      MB LIKE *PB*

      is true.

   ii) The <predicate>

      MB LIKE *PB*

      is true if there exists a partitioning of *MB* into substrings such that:

      1) A substring of *MB* is a sequence of 0 or more contiguous octets of *MB* and each octet of *MB* is part of exactly one substring.

      2) If the *i*-th substring specifier of *PB* is an arbitrary octet specifier, the *i*-th substring of *MB* is any single octet.

3) the *i*-th substring specifier of *PB* is an arbitrary string specifier, then the *i*-th substring of *MB* is any sequence of 0 or more octets.

4) If the *i*-th substring specifier of *PB* is an neither an arbitrary character specifier not an arbitrary string specifier, then the *i*-th substring of *MB* has the same length and bit pattern as that of the substring specifier.

5) The number of substrings of *MB* is equal to the number of substring specifiers of *PB*.

iii) Otherwise:

MB LIKE *PB*

is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <like predicate> shall not be an <octet like predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) The <character match value> shall be a column reference.

   b) A <character pattern> shall be a <value specification>.

   c) An <escape character> shall be a <value specification>.

8.6 <similar predicate>

Function

Specify a character string similarity by means of a regular expression.

Format

```
<similar predicate> ::=
       <match value> [ NOT ] SIMILAR TO <similar pattern>
         [ ESCAPE <escape character> ]

<similar pattern> ::= <character value expression>

<regular expression> ::=
         <regular term>
       | <regular expression> <vertical bar> <regular term>

<regular term> ::=
         <regular factor>
       | <regular term> <regular factor>

<regular factor> ::=
         <regular primary>
       | <regular primary> <asterisk>
       | <regular primary> <plus sign>

<regular primary> ::=
         <character specifier>
       | <percent>
       | <regular character set>
       | <left paren> <regular expression> <right paren>



<character specifier> ::=   <non-escaped character> | <escaped character>

<non-escaped character> ::= !! See the Syntax Rules

<escaped character> ::= !! See the Syntax Rules

<regular character set> ::=
         <underscore>
       | <left bracket> <character enumeration>... <right bracket>
       | <left bracket> <circumflex> <character enumeration>... <right bracket>
       | <left bracket> <colon> <regular character set identifier> <colon> <right bracket>


<character enumeration> ::=
         <character specifier>
       | <character specifier> <minus sign> <character specifier>

<regular character set identifier> ::= <identifier>
```

Syntax Rules

1) The data type of the <match value> shall not be large object string.

2) The data type of the <similar pattern> shall not be large object string.

3) The data type of the <escape character> shall not be large object string.

4) The result of the <character value expression> of <escape character> shall have length 1.

5) The data types of <match value>, <pattern>, and <escape character> shall be character string. <match value>, <pattern>, and <escape character> shall be comparable.

6) The value of the <identifier> that is a <regular character set identifier> shall be either ALPHA, UPPER, LOWER, DIGIT, or ALNUM.

****Editor's Note****
The set of <identifier>s for <regular character set identifier>s could profitable be enlarged to support additional sorts of characters (*e.g.*, ideographs, syllables, *etc.*), as a result of internationalization work subh as that going on in SC22/WG20. See Opportunity **268**.

7) Case:

   a) If <escape character> is not specified, then the collating sequence used for the <similar predicate> is determined by Table 3, "Collating sequence usage for comparisons", taking <match value> as comparand 1 and <similar pattern> as comparand 2.

   b) Otherwise, let *C1* be the coercibility attribute and collating sequence of the <match value>, and *C2* be the coercibility attribute and collating sequence of the <similar pattern>. Let *C3* be the resulting coercibility attribute and collating sequence as determined by Table 2, "Collating coercibility rules for dyadic operators", taking *C1* as the operand 1 coercibility and *C2* as the operand 2 coercibility. The collating sequence used for the <simlar predicate> is determined by Table 3, "Collating sequence usage for comparisons", taking *C3* as the coercibility attribute and collating sequence of comparand 1 and <escape character> as comparand 2.

   It is implementation defined, whether all, some, or no collating sequences other than the default collating sequence for the character set of the <match value> can be used as the collating sequence of the <similar predicate>.

8) A <non-escaped character> is any single character from the character set of the <similar pattern> that is not a <left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore>, <percent>, or the character specified by the result of the <character value expression> of <escape character>. A <character specifier> that is a <non-escaped character> represents itself.

9) An <escaped character> is a sequence of two characters: the character specified by the result of the <character value expression> of <escape character>, followed by a second character that is a <left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore>, <percent>, or the character specified by the result of the <character value expression> of <escape character>. A <character specifier> that is an <escaped character> represents its second character.

10) If the result *E* of the <character value expression> of <escape character> is one of <left bracket>, <right bracket>, <left paren>, <right paren>, <vertical bar>, <circumflex>, <minus sign>, <plus sign>, <asterisk>, <underscore> or <percent>, then *E* shall not be specified in the <regular expression> except as an <escaped character>.

11) If the result of the <character value expression> of <escape character> is a <colon>, then the <regular expression> shall not contain a <regular character set identifier>.

12) A <character enumeration> shall not be specified in a way that both its first and its last <character specifier>s are <non-escaped character>s that are <colon>s.

13) If all of the values of <match value>, <similar pattern>, and <escape character> that do not have the general null class have the same null class, then the result has that null class. Otherwise, the result has the general null class.

Access Rules

None.

General Rules

1) Let *M* be the result of the <character value expression> of the <match value> and let *P* be the result of the <character value expression> of the <similar pattern>.

2) *M* NOT SIMILAR TO *P*

   is equivalent to

   NOT (*M* SIMILAR TO *P*)

3) If the result of the <character value expression> of the <similar pattern> is not an empty string and does not have the format of a <regular expression>, then an exception condition is raised: *data exception—invalid regular expression*.

4) Case:

   a) If ESCAPE is not specified, then if either or both of *M* and *P* are null values, then

      Case:

      i) If the result has the general null class, then the result of

         M SIMILAR TO *P*

         is the general null value.

      ii) Otherwise, the result of

         M SIMILAR TO *P*

         is the minimum of the null values of *M* and *P*.

   b) If ESCAPE is specified, then if one or more of *M*, *P*, and *E* are null values, then

      Case:

      i) If the result has the general null class, then the result of

         M SIMILAR TO *P* ESCAPE *E*

is the general null value.

ii) Otherwise, the result of

M SIMILAR TO *P* ESCAPE *E*

is the minimum of the null values of *M*, *P*, and *E*.

Note: If none of *M*, *P*, and *E* (if present) are null values, then the result is either true or false.

5) The set of characters in a <character enumeration> is defined as

a) If the enumeration is specified in the form "<character specifier> <minus sign> <character specifier>", then the set of all characters that collate greater than or equal to the character represented by the left <character specifier> and less than or equal to the character represented by the right <character specifier>, according to the collating sequence of the pattern *P*.

b) Otherwise, the set of all characters that the <character specifier>s in the <character enumeration> represent.

6) Let *R* be the result of the <character value expression> of the <similar pattern>. The regular language L(*R*) of the <similar pattern> is a (possibly infinite) set of strings. It is defined recursively for well-formed <regular expression>s *Q*, *Q1*, and *Q2* by the following rules:

a) L(*Q1* <vertical bar> *Q2*)

is the union of L(*Q1*) and L(*Q2*)

b) L(*Q* <asterisk>)

is the set of all strings that can be constructed by concatenating zero or more strings from L(*Q*).

c) L(*Q* <plus sign>)

is the set of all strings that can be constructed by concatenating one or more strings from L(*Q*).

d) L(<character specifier>)

is a set that contains a single string of length 1 with the character that the <character specifier> represents

e) L(<percent>)

is the set of all strings of any length (zero or more) from the character set of the pattern *P*.

f) L(<left paren> *Q* <right paren>)

is equal to L(*Q*)

g) L(<underscore>)

is the set of all strings of length 1 from the character set of the pattern *P*.

h) L(<left bracket> <character enumeration> <right bracket>)

is the set of strings of length 1 with character values as defined from the <character enumeration>.

i) L(<left bracket> <circumflex> <character enumeration> <right bracket>)

is the set of strings of length 1 with character values from the character set of the pattern *P* that are not contained in the <character enumeration>.

j) L(<left bracket> <colon> ALPHA <colon> <right bracket>)

is the set of character strings of length 1 that are <simple Latin letter>s.

k) L(<left bracket> <colon> UPPER <colon> <right bracket>)

is the set of character strings of length 1 that are <simple Latin uppercase letter>s.

l) L(<left bracket> <colon> LOWER <colon> <right bracket>)

is the set of character strings of length 1 that are <simple Latin lower case letter>s.

m) L(<left bracket> <colon> DIGIT <colon> <right bracket>)

is the set of character strings of length 1 that are <digit>s.

n) L(<left bracket> <colon> ALNUM <colon> <right bracket>)

is the set of character strings of length 1 that are <simple Latin letter>s or <digit>s.

o) L(*Q1* || *Q2*)

is the set of all strings that can be constructed by concatenating one element of L(*Q1*) and one element of L(*Q2*).

p) L(*Q*)

is the set of the empty string, if *Q* is an empty regular expression.

7) The <similar predicate>

M SIMILAR TO *P*

is true, if there exists at least one element *X* of L(*R*) that is equal to *M* according to the collating sequence of the <similar predicate>; otherwise, it is false.

Note: The <similar predicate> is defined differently from equivalent forms of the LIKE predicate. In particular, blanks at the end of a pattern and collating sequences are handled differently.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall contain no <similar predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.7 <null predicate>

Function

Specify a test for a null value.

Format

```
<null predicate> ::= <row value expression> IS [ NOT ] NULL
           [ <left paren> <null values specification> <right paren> ]

<null values specification> ::= <asterisk> | <null state>
```

Syntax Rules

1) The null class of a <value expression> is defined in Subclause 6.15, "<value expression>".

2) If either no <value expression> of the <row value expression> has a defined null class or any two <value expression>s of the <row value expression> have different defined null classes, then <null state> shall not be specified. Otherwise, <null state> shall be a valid null state of the unique defined null class.

Access Rules

None.

General Rules

1) Let *R* be the value of the <row value expression>.

2) If all the values in *R* are a null value, then "*R* IS NULL" is true; otherwise, it is false.

3) If none of the values in *R* are a null value, then "*R* IS NOT NULL" is true; otherwise, it is false.

Note: For all *R*, "*R* IS NOT NULL" has the same result as "NOT *R* IS NULL" if and only if *R* is of degree 1. Table 15, "<null predicate> semantics", specifies this behavior.

Table 15—<null predicate> semantics

Expression	*R* IS NULL	*R* IS NOT NULL	NOT *R* IS NULL	NOT *R* IS NOT NULL
degree 1: null	*true*	*false*	*false*	*true*
degree 1: not null	*false*	*true*	*true*	*false*
degree > 1: all null	*true*	*false*	*false*	*true*
degree > 1: some null	*false*	*false*	*true*	*true*
degree > 1: none null	*false*	*true*	*true*	*false*

4) Case:

   a) If <null values specification> is not specified, then:

      i) "*R* IS NULL" is true if and only if all the values in *R* are the general null value; otherwise, it is false.

      ii) "*R* IS NOT NULL" is true if and only if all the values in *R* are non-null or have a null value that is not the general null value; otherwise, it is false.

   b) If <null values specification> is specified as "*", then:

      i) "*R* IS NULL(*)" is true if and only if all the values in *R* are some null value; otherwise, it is false.

      ii) "*R* IS NOT NULL(*)" is true if and only if all the values in *R* are non-null; otherwise, it is false.

   c) If <null values specification> is specified as "<null state>", then:

      i) Let *C* be the null state specified by <null state>.

      ii) "*R* IS NULL (*C*)" is true if and only if all the values in *R* are a null value with a null state of *C*; otherwise, it is false.

      iii) "*R* IS NOT NULL (*C*)" is true if and only if all the values in *R* are non-null or have a null value with a null state other than that of *C*; otherwise, it is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <null predicate> shall contain no <null values specification>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <row value expression> shall be a column reference.

8.8 <quantified comparison predicate>

Function

Specify a quantified comparison.

Format

```
<quantified comparison predicate> ::=
      <row value expression> <comp op> <quantifier> <table subquery>

<quantifier> ::= <all> | <some>

<all> ::= ALL

<some> ::= SOME | ANY
```

Syntax Rules

1) The <row value expression> shall be of the same degree as the result of the <table subquery>.

2) The data types of the values of the <row value expression> shall be respectively comparable to those of the corresonding columns of the <table subquery>.

3) The data types of every value of the <row value constructor> shall not be large object string.

4) The collating sequence for each pair of respective values in the <quantified comparison predicate> is determined in the same manner as described in Subclause 8.2, "<comparison predicate>".

Access Rules

None.

General Rules

1) Let *R* be the result of the <row value expression> and let *T* be the result of the <table subquery>.

2) The result of "*R* <comp op> <quantifier> *T*" is derived by the application of the implied <comparison predicate> "*R* <comp op> *RT*" to every row *RT* in *T*:

   Case:

   a) If *T* is empty or if the implied <comparison predicate> is true for every row *RT* in *T*, then "*R* <comp op> <all> *T*" is true.

   b) If the implied <comparison predicate> is false for at least one row *RT* in *T*, then "*R* <comp op> <all> *T*" is false.

   c) If the implied <comparison predicate> is true for at least one row *RT* in *T*, then "*R* <comp op> <some> *T*" is true.

d) If *T* is empty or if the implied <comparison predicate> is false for every row *RT* in *T*, then "*R* <comp op> <some> *T*" is false.

e) If "*R* <comp op> <quantifier> *T*" is neither true nor false, then it is the general null value.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.9 <exists predicate>

Function

Specify a test for a non-empty set.

Format

```
<exists predicate> ::= EXISTS <table subquery>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the result of the <table subquery>.

2) If the cardinality of *T* is greater than 0, then the result of the <exists predicate> is true; otherwise, the result of the <exists predicate> is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.10 <unique predicate>

Function

Specify a test for the absence of duplicate rows.

Format

```
<unique predicate> ::= UNIQUE <table subquery>
```

Syntax Rules

1) The data type of every column of *T* shall not be large object string.

Access Rules

None.

General Rules

1) Let *T* be the result of the <table subquery>.

2) If there are no two rows in *T* such that the value of each column in one row is non-null and is equal to the value of the corresponding column in the other row according to Subclause 8.2, "<comparison predicate>", then the result of the <unique predicate> is true; otherwise, the result of the <unique predicate> is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <unique predicate>.

8.11 <match predicate>

Function

Specify a test for matching rows.

Format

```
<match predicate> ::=
      <row value expression> MATCH [ UNIQUE ] [ PARTIAL | FULL ] <table subquery>
```

Syntax Rules

1) The <row value expression> shall be of the same degree as the <table subquery>.

2) The data types of the values of the <row value expression> shall be respectively comparable to those of the corresponding columns of the <table subquery>.

3) The data type of every value of the <row value constructor> and of every column of the <table subquery> shall not be large object string.

4) The collating sequence for each pair of respective values in the <match predicate> is determined in the same manner as described in Subclause 8.2, "<comparison predicate>".

Access Rules

None.

General Rules

1) Let R be the <row value expression>.

2) If neither PARTIAL nor FULL is specified, then

   Case:

   a) If some value in R is a null value, then the <match predicate> is true.

   b) If no value in R is a null value, then

      Case:

      i) If UNIQUE is not specified and there exists a (possibly non-unique) row RT_i of the <table subquery> such that

         $R = RT_i$

         then the <match predicate> is true.

      ii) If UNIQUE is specified and there is a unique row RT_i of the <table subquery> such that

         $R = RT_i$

         then the <match predicate> is true.

iii) Otherwise, the <match predicate> is false.

3) If PARTIAL is specified, then

Case:

a) If all values in R are a null value, then the <match predicate> is true.

b) Otherwise,

Case:

i) If UNIQUE is not specified and there exists a (possibly non-unique) row RT_i of the <table subquery> such that each non-null value of R equals its corresponding value in RT_i, then the <match predicate> is true.

ii) If UNIQUE is specified and there is a unique row RT_i of the <table subquery> such that each non-null value of R equals its corresponding value in RT_i, then the <match predicate> is true.

iii) Otherwise, the <match predicate> is false.

4) If FULL is specified, then

Case:

a) If all values in R are a null value, then the <match predicate> is true.

b) If no values in R are a null value, then

Case:

i) If UNIQUE is not specified and there exists a (possibly non-unique) row RT_i of the <table subquery> such that

$R = RT_i$

then the <match predicate> is true.

ii) If UNIQUE is specified and there exists a unique row RT_i of the <table subquery> such that

$R = RT_i$

then the <match predicate> is true.

iii) Otherwise, the <match predicate> is false.

c) Otherwise, the <match predicate> is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <match predicate>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.12 <overlaps predicate>

Function

Specify a test for an overlap between two events.

Format

```
<overlaps predicate> ::=
      <row value expression 1> OVERLAPS <row value expression 2>

<row value expression 1> ::= <row value expression>

<row value expression 2> ::= <row value expression>
```

Syntax Rules

1) The degrees of <row value expression 1> and <row value expression 2> shall both be 2.

2) The data type of the first column of <row value expression 1> and the first column of <row value expression 2> shall both be datetime data types and these data types shall be comparable.

   Note: Two datetimes are comparable only if they have the same <datetime field>s; see Subclause 4.9.1, "Datetimes".

3) The data type of the second column of each <row value expression> shall be a datetime data type or INTERVAL.

   Case:

   a) If the data type is INTERVAL, then the precision of the data type shall be such that the interval can be added to the datetime data type contained in the first column of the <row value expression>.

   b) If the data type is a datetime data type, then it shall be comparable with the datetime data type contained in the first column of the <row value expression>.

Access Rules

None.

General Rules

1) Let *D1* be the value of the first column of <row value expression 1> and *D2* be the value of the first column of <row value expression 2>.

2) Case:

   a) If the data type of the second column of <row value expression 1> is a datetime data type, then let *E1* be the value of the second column of <row value expression 1>.

   b) If the data type of the second column of <row value expression 1> is INTERVAL, then let *I1* be the value of the second column of <row value expression 1>. Let $E1 = D1 + I1$.

3) If *D1* is the null value or if *E1* < *D1*, then let *S1* = *E1* and let *T1* = *D1*. Otherwise, let *S1* = *D1* and let *T1* = *E1*.

4) Case:

   a) If the data type of the second column of <row value expression 2> is a datetime data type, then let *E2* be the value of the second column of <row value expression 2>.

   b) If the data type of the second column of <row value expression 2> is INTERVAL, then let *I2* be the value of the second column of <row value expression 2>. Let *E2* = *D2* + *I2*.

5) If *D2* is the null value or if *E2* < *D2*, then let *S2* = *E2* and let *T2* = *D2*. Otherwise, let *S2* = *D2* and let *T2* = *E2*.

6) The result of the <overlaps predicate> is the result of the following expression:

   (*S1* > *S2* AND NOT (*S1* >= *T2* AND *T1* >= *T2*))
   OR
   (*S2* > *S1* AND NOT (*S2* >= *T1* AND *T2* >= *T1*))
   OR
   (*S1* = *S2* AND (*T1* <> *T2* OR *T1* = *T2*))

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

8.13 <quantified predicate>

Function

Specify a quantified predicate.

Format

```
<quantified predicate> ::=
        <existential clause> <left paren> <search condition> <right paren>
      | <universal clause> <left paren> <search condition> <right paren>
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
      | <quantified comparison predicate>
```

```
<existential clause> ::=
      FOR <some> <table reference list>

<universal clause> ::=
      FOR <all> <table reference list>
```

- 1 production deleted.

Syntax Rules

None.

Access Rules

1) Let *T* be any table that is referenced in an <existential clause> or <universal clause>. If *T* is not a declared temporary table or a temporary view, then then the applicable privileges shall include SELECT for at least one column of *T*.

General Rules

- 1 Rule deleted.

1) Let *T* be the result of the <table reference list>.

   Case:

   a) If the <quantified predicate> immediately contains an <existential clause>, then the result of the <quantified predicate> is:

      i) *false* if *T* is empty or if the <search condition> is false for every row of *T*;

      ii) *true* if there exists any row of *T* for which the <search condition> is true;

      iii) Otherwise, the general null value.

b) If the <quantified predicate> immediately contains a <universal clause>, then the result of the <quantified predicate> is:

i) *true* if *T* is empty or if the <search condition> is true for every row of *T*;

ii) *false* if there exists any row of *T* for which the <search condition> is false;

iii) Otherwise, the general null value.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall contain no <quantified predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

ANSI Only—caused by ISO changes not yet considered by ANSI

8.14 <there is predicate>

Function

Specify an existentially quantified predicate.

Format

```
<there is predicate> ::=
       <left paren> <there is clause> <where clause> <right paren>

<there is clause> ::=
       THERE IS <table reference list>
```

Syntax Rules

None.

Access Rules

None.

General Rules

- 1 Rule deleted.

1) Let *T* be the result of the <table reference list>. The result of the <there is predicate> is:

   a) false if *T* is empty or if the <where clause> is false for every row of *T*.

   b) true if there exists any row of *T* for which the <where clause> is true.

   c) unknown otherwise.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <there is predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.15 <distinct predicate>

Function

Specify a test of whether two row values are distinct

Format

```
<distinct predicate> ::=
      <row value expression 3> IS DISTINCT FROM <row value expression 4>

<row value expression 3> ::= <row value expression>

<row value expression 4> ::= <row value expression>
```

Syntax Rules

1) The two <row value expression>s shall be of the same degree.

2) Let *respective values* be values with the same ordinal position.

3) The data type of every respective value shall not be large object string.

4) The data types of the respective values of the two <row value expression>s shall be comparable.

5) Let *X* be a value in the first <row value expression> and let *Y* be the respective value in the second <row value expression>.

Access Rules

None.

General Rules

1) The result of the <distinct predicate> is either true or false.

2) Two <row value expression>s are distinct if, for any pair of respective values *X* and *Y*, *X* IS DISTINCT FROM *Y* is true.

3) Case:

   a) If the data type of *X* or *Y* is a set or multiset type, then "*X* IS DISTINCT FROM *Y*" is effectively computed as follows:

      i) Let *TEMPX* be a copy of *X*; let *TEMPY* be a copy of *Y*.

      ii) For each element of *TEMPX*, if that element is not distinct from an element in *TEMPY*, then delete that element from *TEMPX* and that element in *TEMPY*.

      iii) If *TEMPX* is empty and *TEMPY* is empty, then "*X* IS DISTINCT FROM *Y*" is false. Otherwise, "*X* IS DISTINCT FROM *Y*" is true.

b) If the data type of *X* or *Y* is a list type, then "*X* IS DISTINCT FROM *Y*" is effectively computed as follows:

i) Let *NX* be the number of elements in *X*; let *NY* be the number of elements in *Y*.

ii) Let EX_i be the *i*-th element of *X*; let EY_i be the *i*-th element of *Y*.

iii) If *NX* is not equal to *NY*, then "*X* IS DISTINCT FROM *Y*" is true.

iv) If *NX* equals zero and *NY* equals zero, then "*X* IS DISTINCT FROM *Y*" is false.

v) If "EX_i IS DISTINCT FROM EY_i" is false for all *i* from *i=1* to *i=NX*, then "*X* IS DISTINCT FROM *Y*" is false.

vi) Otherwise, "*X* IS DISTINCT FROM *Y*" is true.

c) Otherwise,

Case:

i) "*X* IS DISTINCT FROM *Y*" is false if either:

1) *X* and *Y* are the same null value, or

2) *X* = *Y* according to Subclause 8.2, "<comparison predicate>".

ii) Otherwise, "*X* IS DISTINCT FROM *Y*" is true.

4) If two <row value expression>s are not distinct, then they are said to be *duplicates*. If a number of <row value expression>s are all duplicates of each other, then all except one are said to be *redundant duplicates*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <distinct predicate>.

8.16 <boolean predicate>

Function

Specify a test for whether a boolean value is *true* , *false* , or *unknown* .

Format

```
<boolean predicate> ::=
      <boolean value expression> [ IS [ NOT ] <truth value> ]

<truth value> ::=
        TRUE
      | FALSE
      | UNKNOWN
```

Syntax Rules

1) If NOT is specified in a <boolean predicate>, then let *BV* be the contained <boolean value expression> and let *TV* be the contained <truth value>. The <boolean predicate> is equivalent to:

   (NOT (*BV* IS *TV*))

Access Rules

None.

General Rules

1) Let *R* be the value of *BV* and let *T* be the value of *TV*.

2) The result of *BV* IS *TV* is derived by the application of the boolean operator “IS” to the result derived from the evaluation of *BV*.

3) “IS” is defined by Table 16, "Truth table for the IS boolean".

Table 16—Truth table for the IS boolean

IS	TRUE	FALSE	UNKNOWN
true	*true*	*false*	*false*
false	*false*	*true*	*false*
null	*false*	*false*	*true*

Note: The result of a <boolean predicate> is always either *true* or *false* .

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Intermediate SQL language shall not specify a <boolean predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

8.17 <type predicate>

Function

Specify a type test.

Format

```
<type predicate> ::=
       TYPE <type predicate clause> [ IS ] IN <type predicate set>

<type predicate clause> ::=
         <expression type>
       | <data type>

<type predicate set> ::=
       <left paren> <type predicate list> <right paren>

<expression type> ::= <left paren> <value expression> <right paren>

<type predicate list> ::=
       <type predicate element> [ { <comma> <type predicate element> }... ]

<type predicate element> ::=
         <type predicate clause>
       | ONLY <type predicate clause>
       | ALL <type predicate clause>
             [ <left paren> EXCEPT <type predicate list> <right paren> ]
```

Syntax Rules

1) Let *VE* be the <value expression> immediately contained in <expression type>. Let *CDT* be the data type of *VE*.

2) Let *DTF* be the subtype family of *CDT*.

3) Let *TP* be a <type predicate>. Let *TPC* be the <type predicate clause> immediately contained in *TP*. Let *TPS* be the <type predicate set) immediately contained in *TP*.

Access Rules

None.

General Rules

1) If <expression type> is specified, then:

   Case:

   a) If *VE* is null, then let *EDT* be NONE.

   b) Otherwise, let *EDT* be the type *T* of *VE*.

   Note: *EDT* specifies a type (or subtype) contained in *DTF* or specified NONE.

2) A <type predicate clause> *TPC* specifies some data type.

   Case:

   a) If <expression type> is specified, then the type of *TPC* is *EDT*.

   b) Otherwise, the type of *TPC* is the data type specified by <data type>.

3) A <type predicate element> *TPE* specifies a set of data types. Let *DT1* be the data type specified by the immediately contained <type predicate clause>. Let *DTF1* be the subtype family of *DT1*.

   Case:

   a) If ONLY is specified, then *TPE* specifies the singleton set of *DT1*.

   b) If neither ALL or ONLY is specified, then *TPE* specifies the set of all data types in *DTF1*.

   c) If EXCEPT is specified, then *TPE* specifies the set difference of the set of all data types in *DTF1* with the set of data types specified by the immediately contained <table predicate list>.

4) A <type predicate list> specifies the set union of all the data type sets specified by each of the immediately contained <type predicate element>s.

5) Let *DT* be the data type specified by the <type predicate clause> of *TP*. Let *SDT* be the set of data types specified by the <type predicate set> of *TP*.

   Case:

   a) If *DT* is NONE, then *TP* is unknown.

   b) If *DT* appears in *SDT*, then *TP* is true.

   c) Otherwise, *TP* is false.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain a <type predicate>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.18 <search condition>

Function

Specify a condition that is true, false, or unknown, depending on the value of a <boolean value expression>.

Format

```
<search condition> ::=
      <boolean value expression>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) When a <search condition> *S* is evaluated against a row of a table, each reference to a column of that table by a column reference directly contained in *S* is a reference to the value of that column in that row.

2) Case:

   a) If the result of evaluating the <boolean value expression> is null, then the result of the <search condition> is unknown.

   b) Otherwise, the result of the <search condition> is the result of the <boolean value expression>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <boolean primary> shall not specify a <boolean value expression>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <boolean test> shall not specify a <truth value>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

9 Data assignment rules and function determination

9.1 Retrieval assignment

Function

Specify rules for value assignments that retrieve SQL-data.

Syntax Rules

1) Let *T* and *V* be a *TARGET* and *VALUE* specified in an application of this Subclause.

2) If the data type of *T* is character string or character large object locator, bit string, binary string or binary large object locator, numeric, an enumerated type, boolean, datetime, interval, a collection type, an abstract data type, or a distinct type, then the data type of *V* shall be a mutually assignable character string type, a bit string type, a binary string type, a numeric type, the same enumerated type, boolean, the same datetime type, a comparable interval type, the same collection type, a subtype of the data type of *T*, or the same distinct type, respectively.

   Note: "subtype" is defined in Subclause 4.11.5, "Subtypes and supertypes for ADTs".

3) If the data type of *T* is an abstract data type, then the Syntax Rules of Subclause 9.4, "Abstract data type assignment", are applied to *T* and *V* as *TARGET* and *VALUE*, respectively.

ANSI Only–SQL3

4) If the data type of the target of a value assignment is a row identifier data type referencing the subtable family rooted by table *T*, then the assigned value shall have a row identifier data type from the subtable family rooted in *T*.

5) If the data type of *T* is a row type, then:

   a) The data type of *V* shall be a row type.

   b) The degree of *V* shall be the same as the degree of *T*.

   c) The Syntax Rules of this Subclause apply to the *i*-th element of *T* and the *i*-th element of *V*, as *TARGET* and *VALUE*, respectively.

General Rules

1) If *V* is a null value and *T* is an externally-supplied parameter, then

   Case:

   a) If an indicator is specified for *T*, then that indicator is set to

Case:

i) If the null value is the general null value, then −1.

ii) Otherwise, the negative of the position number of the null state.

b) If no indicator is specified for *T*, then an exception condition is raised: *data exception—null value, no indicator parameter*.

2) If *V* is a null value and *T* is not an an externally-supplied parameter, then *T* is set to *V*.

3) If *V* is not a null value and *T* has an indicator, then

Case:

a) If the data type of *T* is character string or character large object locator, bit string, or binary string or binary large object locator and the length in characters, bits, or octets, respectively, *M* of *V* is greater than the length in characters, bits, or octets, respectively, of *T*, then the indicator is set to *M*. If *M* exceeds the maximum value that the indicator can contain, then an exception condition is raised: *data exception—indicator overflow*.

b) Otherwise, the indicator is set to 0.

4) If *V* is not a null value, then

Case:

a) If the data type of *T* is fixed-length character string with length in characters *L* and the length in characters of *V* is equal to *L*, then the value of *T* is set to *V*.

b) If the data type of *T* is fixed-length character string with length in characters *L*, and the length in characters of *V* is greater than *L*, then the value of *T* is set to the first *L* characters of *V* and a completion condition is raised: *warning—string data, right truncation*.

c) If the data type of *T* is fixed-length character string with length in characters *L*, and the length in characters *M* of *V* is smaller than *L*, then the first *M* characters of *T* are set to *V*, and the last *L*−*M* characters of *T* are set to <space>s.

d) If the data type of *T* is variable-length character string and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

e) If the data type of *T* is variable-length character string and the length in characters of *V* is greater than the maximum length in characters *L* of *T*, then the value of *T* is set to the first *L* characters of *V*, the length in characters of *T* becomes *L*, and a completion condition is raised: *warning—string data, right truncation*.

f) If the data type of *T* is character large object string and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

g) If the data type of *T* is character large object string and the length in characters of *V* is greater than the maximum length in characters *L* of *T*, then the value of *T* is set to the first *L* characters of *V*, the length in characters of *T* becomes *L*, and a completion condition is raised: *warning—string data, right truncation*.

h) If the data type of *T* is character large object locator, then a locator *H* that uniquely identifies *V* is generated and the value of *T* is set to an implementation-dependent character string value that represents *H*.

i) If the data type of *T* is fixed-length bit string with length in bits *L* and the length in bits of *V* is equal to *L*, then the value of *T* is set to *V*.

j) If the data type of *T* is fixed-length bit string with length in bits *L* and the length in bits of *V* is greater than *L*, then the value of *T* is set to the first *L* bits of *V* and a completion condition is raised: *warning—string data, right truncation*.

k) If the data type of *T* is fixed-length bit string with length in bits *L* and the length in bits *M* of *V* is smaller than *L*, then the first *M* bits of *T* are set to *V*, the remaining bits of *T* are set to bits each with the value of 0, and a completion condition is raised: *warning—implicit zero-bit padding*.

l) If the data type of *T* is variable-length bit string and the length in bits *M* of *V* is not greater than the maximum length in bits of *T*, then the value of *T* is set to *V* and the length in bits of *T* is set to *M*.

m) If the data type of *T* is variable-length bit string, and the length in bits of *V* is greater than the maximum length in bits *L* of *T*, then the value of *T* is set to the first *L* bits of *V*, the length in bits of *T* is set to *L*, and a completion condition is raised: *warning—string data, right truncation*.

n) If the data type of *T* is binary string and the length in octets *M* of *V* is not greater than the maximum length in octets of *T*, then the value of *T* is set to *V* and the length in octets of *T* is set to *M*.

o) If the data type of *T* is binary string and the length in octets of *V* is greater than the maximum length in octets *L* of *T*, then the value of *T* is set to the first *L* octets of *V*, the length in octets of *T* becomes *L*, and a completion condition is raised: *warning—string data, right truncation*.

p) If the data type of *T* is binary large object locator, then, a locator *H* that uniquely identifies *V* is generated and the value of *T* is set to an implementation-dependent character string value that represents *H*.

q) If the data type of *T* is numeric, then

Case:

i) If *V* is a member of the data type of *T*, then *T* is set to *V*.

ii) If a member of the data type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. If the data type of *T* is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

iii) Otherwise, an exception condition is raised: *data exception—numeric value out of range*.

r) If the data type of *T* is datetime and there is a representation of *V* in the data type of *T*, then the value of *T* is set to that representation.

s) If the data type of *T* is interval, then

Case:

i) If *V* is a member of the data type of *T*, then *T* is set to *V*.

ii) If a member of the data type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. It is implementation-defined whether the approximation is obtained by rounding or by truncation.

iii) Otherwise, an exception condition is raised: *data exception—interval field overflow*.

t) If the data type of *T* is a row type, then the General Rules of this Subclause are applied to the *i*-th element of *T* and the *i*-th element of *V* as *TARGET* and *VALUE*, respectively.

u) If the data type of *T* is an abstract data type, then the General Rules of Subclause 9.4, "Abstract data type assignment", are applied to *T* and *V* as *TARGET* and *VALUE*, respectively.

v) If the data type of *T* is a distinct type, then the value of *T* is set to *V*.

w) If the data type of *T* is a collection type, then the value of *T* is set to *V*.

9.2 Store assignment

Function

Specify rules for value assignments that store SQL-data or store into external function parameters.

Syntax Rules

1) Let *T* and *VO* be a *TARGET* and *VALUE* specified in an application of this Subclause. If the data type of *VO* is character large object locator or binary large object locator, then let *V* be the value identified by *VO*. otherwise, let *V* be *VO*.

2) If *V* is specified by NULL (<null state>), then <null state> shall identify a null state in the null class of *T*.

3) If the data type of *T* is character string, bit string, binary string, numeric, an enumerated type, boolean, datetime, interval, a collection type, an abstract data type, or a distinct type, then the data type of *V* shall be character string, bit string, binary string, numeric, the same enumerated type, boolean, the same datetime type, a comparable interval type, the same collection type, or a subtype of the data type of *T*, or the same distinct type, respectively.

   Note: "subtype" is defined in Subclause 4.11.5, "Subtypes and supertypes for ADTs".

4) If the data type of *T* is a row type, then:

   a) The data type of *V* shall be a row type.

   b) The degree of *V* shall be the same as the degree of *T*.

   c) The Syntax Rules of this Subclause apply to the *i*-th element of *T* and the *i*-th element of *V*, as *TARGET* and *VALUE*, respectively.

5) If the data type of *T* is an abstract data type, then the Syntax Rules of Subclause 9.4, "Abstract data type assignment", are applied to *T* and *V* as *TARGET* and *VALUE*, respectively.

ANSI Only–SQL3

6) If the data type of the target of a value assignment is a row identifier data type referencing the subtable family rooted by table *T*, then the assigned value shall have a row identifier data type from the subtable family rooted in *T*.

General Rules

1) Case:

   a) If *V* is a null value, then

      Case:

      i) If *V* is specified using NULL (<null state>), then *T* is set to the null state specified.

ii) If *V* is a host parameter reference and contains an indicator parameter, then

Case:

1) If the value of the indicator parameter is equal to –1, then *T* is set to the general null value.

2) If the value of the indicator parameter is equal to the negative of the position of a null state in the null class of *T*, then *T* is set to that null state.

3) Otherwise, an exception condition is raised: *data exception—invalid indicator parameter value*.

iii) Otherwise, *T* is set to the general null value.

b) Otherwise,

Case:

i) If the data type of *T* is fixed-length character string with length in characters *L* and the length in characters of *V* is equal to *L*, then the value of *T* is set to *V*.

ii) If the data type of *T* is fixed-length character string with length in characters *L* and the length in characters *M* of *V* is larger than *L*, then

Case:

1) If the rightmost *M*–*L* characters of *V* are all <space>s, then the value of *T* is set to the first *L* characters of *V*.

2) If one or more of the rightmost *M*–*L* characters of *V* are not <space>s, then an exception condition is raised: *data exception—string data, right truncation*.

iii) If the data type of *T* is fixed-length character string with length in characters *L* and the length in characters *M* of *V* is less than *L*, then the first *M* characters of *T* are set to *V* and the last *L*–*M* characters of *T* are set to <space>s.

iv) If the data type of *T* is variable-length character string and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

v) If the data type of *T* is variable-length character string and the length in characters *M* of *V* is greater than the maximum length in characters *L* of *T*, then,

Case:

1) If the rightmost *M*–*L* characters of *V* are all <space>s, then the value of *T* is set to the first *L* characters of *V* and the length in characters of *T* is set to *L*.

2) If one or more of the rightmost *M*–*L* characters of *V* are not <space>s, then an exception condition is raised: *data exception—string data, right truncation*.

vi) If the data type of *T* is character large object string and the length in characters *M* of *V* is not greater than the maximum length in characters of *T*, then the value of *T* is set to *V* and the length in characters of *T* is set to *M*.

vii) If the data type of *T* is character large object string and the length in characters *M* of *V* is greater than the maximum length in characters *L* of *T*, then

Case:

i) If the rightmost $M - L$ characters of V are all <space>s, then the value of T is set to the first L characters of V and the length in characters of T is set to L.

ii) If one or more of the rightmost $M - L$ characters of V are not <space>s, then an exception condition is raised: *data exception—string data, right truncation*.

viii) If the data type of T is fixed-length bit string with length in bits L and the length in bits of V is equal to L, then the value of T is set to V.

ix) If the data type of T is fixed-length bit string with length in bits L and the length in bits M of V is greater than L, then an exception condition is raised: *data exception—string data, right truncation*.

x) If the data type of T is fixed-length bit string with length in bits L and the length in bits M of V is less than L, then an exception condition is raised: *data exception—string data, length mismatch*.

xi) If the data type of T is variable-length bit string and the length in bits M of V is not greater than the maximum length in bits of T, then the value of T is set to V and the length in bits of T is set to M.

xii) If the data type of T is variable-length bit string, and the length in bits M of V is greater than the maximum length in bits L of T, then an exception condition is raised: *data exception—string data, right truncation*.

xiii) If the data type of T is binary string and the length in octets M of V is not greater than the maximum length in octets of T, then the value of T is set to V and the length in octets of T is set to M.

xiv) If the data type of T is binary string and the length in octets M of V is greater than the maximum length in octets L of T, then

Case:

i) If the rightmost $M - L$ octets of V are all equal to X'00', then the value of T is set to the first L octets of V and the length in octets of T is set to L.

ii) If one or more of the rightmost $M - L$ octets of V are not equal to X'00', then an exception condition is raised: *data exception—string data, right truncation*.

xv) If the data type of T is numeric, then

Case:

1) If V is a member of the data type of T, then T is set to V.

2) If a member of the data type of T can be obtained from V by rounding or truncation, then T is set to that value. If the data type of T is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

3) Otherwise, an exception condition is raised: *data exception—numeric value out of range*.

xvi) If the data type of *T* is datetime and there is a representation of *V* in the data type of *T*, then the value of *T* is set to that representation.

xvii) If the data type of *T* is interval, then

Case:

1) If *V* is a member of the data type of *T*, then *T* is set to *V*.

2) If a member of the data type of *T* can be obtained from *V* by rounding or truncation, then *T* is set to that value. It is implementation-defined whether the approximation is obtained by rounding or by truncation.

3) Otherwise, an exception condition is raised: *data exception—interval field overflow*.

xviii) If the data type of *T* is boolean, then the value of *T* is set to *V*.

xix) If the data type of *T* is an enumerated type, then the value of *T* is set to the value of *V*.

xx) If the data type of *T* is a row type, then the General Rules of this Subclause are applied to the *i*-th element of *T* and the *i*-th element of *V* as *TARGET* and *VALUE*, respectively.

xxi) If the data type of *T* is an abstract data type, then the General Rules of Subclause 9.4, "Abstract data type assignment", are applied to *T* and *V* as *TARGET* and *VALUE*, respectively.

xxii) If the data type of *T* is a distinct type, then the value of *T* is set to *V*.

xxiii) If the data type of *T* is a collection type, then the value of *T* is set to *V*.

2) If *T* is a column and the column definition of *T* includes the name of a domain whose domain descriptor includes a domain constraint *D*, then *D* is effectively checked. If *D* is not satisfied, then an exception condition is raised: *integrity constraint violation*.

9.3 Set operation result data types and nullabilities

Function

Specify the Syntax Rules and result data types and nullabilities for <case expression>s, <collection value expression>s, and <query expression>s having set operators.

Syntax Rules

1) Let *DTS* be a set of data types specified in an application of this Subclause.

2) All of the data types in *DTS* shall be comparable.

3) Case:

   a) If any of the data types in *DTS* is character string, then all data types in *DTS* shall be character string, and all of them shall have the same character repertoire. That character repertoire is the character repertoire of the result. The character set of the result is the character set of one of the data types in *DTS*. The specific character set chosen is implementation-dependent. The collating sequence and the coercibility attribute are determined as specified in Table 2, "Collating coercibility rules for dyadic operators".

      Case:

      i) If any of the data types in *DTS* is character large object string, then the result data type is character large object string with maximum length in characters equal to the maximum of the lengths in characters and maximum lengths in characters of the data types in *DTS*.

      ii) If any of the data types in *DTS* is variable-length character string, then the result data type is variable-length character string with maximum length in characters equal to the maximum of the lengths in characters and maximum lengths in characters of the data types in *DTS*.

      iii) Otherwise, the result data type is fixed-length character string with length in characters equal to the maximum of the lengths in characters of the data types in *DTS*.

   b) If any of the data types in *DTS* is bit string, then all data types in *DTS* shall be bit string.

      Case:

      i) If any of the data types in *DTS* is variable-length bit string, then the result data type is variable-length bit string with maximum length in bits equal to the maximum of the lengths in bits and maximum lengths in bits of the data types in *DTS*.

      ii) Otherwise, the result data type is fixed-length bit string with length in bits equal to the maximum of the lengths in bits of the data types in *DTS*.

   c) If any of the data types in *DTS* is binary string, then the result data type is binary string with maximum length in octets equal to the maximum of the lengths in octets and maximum lengths in octets of the data types in *DTS*.

   d) If all of the data types in *DTS* are exact numeric, then the result data type is exact numeric with implementation-defined precision and with scale equal to the maximum of the scales of the data types in *DTS*.

e) If any data type in *DTS* is approximate numeric, then each data type in *DTS* shall be numeric and the result data type is approximate numeric with implementation-defined precision.

f) If any data type in *DTS* is a datetime data type, then each data type in *DTS* shall be the same datetime data type. The result data type is the same datetime data type.

g) If any data type in *DTS* is interval, then each data type in *DTS* shall be interval. If the precision of any data type in *DTS* specifies YEAR or MONTH, then the precision of each data type shall specify only YEAR or MONTH. If the precision of any data type in *DTS* specifies DAY, HOUR, MINUTE, or SECOND(*N*), then the precision of no data type of *DTS* shall specify the <datetime field>s YEAR and MONTH. The result data type is interval with precision "*S* TO *E*", where *S* and *E* are the most significant of the <start field>s and the least significant of the <end field>s of the data types in *DTS*, respectively.

h) If any data type in *DTS* is an enumerated type, then each data type in *DTS* shall be that enumerated type. The result data type is that enumerated type.

i) If any data type in *DTS* is boolean, then each data type in *DTS* shall be boolean. The result data type is boolean.

j) If any data type in *DTS* is a row type, then each data type in *DTS* shall be a row type with the same degree and the data type of each field in the same ordinal position of every row type shall be comparable. The result data type is a row defined by an ordered sequence of (<field name>, data type) pairs, FD_i, where <field name> is implementation-defined and data type is the data type resulting from the application of this Subclause to the set of data types of each fields in the same ordinal position as FD_i in every row type in *DTS*.

k) If any data type in *DTS* is an abstract data type, then each data type in *DTS* shall be an abstract data type that is in the same subtype family. Let *ADT* be the data type that is the most specific unique type that is a supertype of each data type in *DTS*. The result data type is *ADT*.

Note: *Most specific unique type* is defined in Subclause 4.11.5, "Subtypes and supertypes for ADTs".

l) If any data type in *DTS* is a distinct type, then each data type in *DTS* shall be the same distinct type. The result data type is that distinct type.

m) If any data type in *DTS* is a collection type, then each data type in *DTS* shall be the same collection type.

4) If all of the null classes of the data types in *DTS* that are not the general null class are the same null class, then the result null class is that null class. Otherwise, the result has the general null class.

ANSI Only–SQL3

5) If any data type in *DTS* is a row identifier data type, then each data type in *DTS* shall be a row identifier data type. Let *RIDT* be the most specific unique row identifier data type that is a supertype of each data type in *DTS*. The result data type is *RIDT*.

General Rules

None.

9.4 Abstract data type assignment

Function

Specify rules for assignments of abstract data types.

Syntax Rules

1) Let *T* and *V* be a *TARGET* and *VALUE* specified in an application of this Subclause.

General Rules

1) If *T* is a value ADT, then the value of *T* is set to the value of *V*.

ANSI Only–SQL3

For each item that is part of the new value of *T* whose data type is an object abstract data type and whose declaration specifies INSTANCE, a new object identifier is geenrated and associated with that new value.

2) If *T* is an object ADT, then

Case:

a) If the declaration of *T* specifies INSTANCE, then the value of *T* is set to the value of *V*.

ANSI Only–SQL3

For each item that is part of the new value of *T* whose data type is an object abstract data type and whose declaration specifies INSTANCE, a new object identifier is geenrated and associated with that new value.

b) Otherwise, the value of *T* is set to the value of the OID attribute of
ANSI *V* associated with *V*.
ISO *V*.

9.5 Subject type determination

Function

Determine the type applicable to a given unqualified type reference.

Syntax Rules

1) Let *TN* be the type name and let *TPL* be the <template parameter list>, if any, used in an application of this Subclause.

2) If *TN* does not contain a <schema name>, then:

   a) If *TN* is contained in a <module> without an intervening <schema definition>, then let *DP* be the default SQL-path of the <module>. If *TN* is contained in a <schema definition> without an intervening <module>, then let *DP* be the default path of the <schema definition>.

   b) Let S_i be the *i*-th <schema name> in *DP*. Let *candidate types* be the set of types resulting from the union of all possibly candidate types of *TN* in every S_i.

      Note: "possibly candidate type" is defined in Subclause 5.4, "Names and identifiers".

3) If *TN* contains a <schema name> *SNT*, then let the candidate types by the set of possibly candidate types contained in the schema whose <schema name> is equal to *SNT*.

4) Case:

   i) If *TN* is a type template, then the Syntax Rules of Subclause 8.1, "Subject routine determination", in Part 4 of this
      ANSI American
      ISO International
      Standard, replacing "function" with "type template", are applied to the set of candidate types and operations of *TPL*, yielding a set of subject types.

   ii) Otherwise, let the set of subject types be the candidate types.

5) Case:

   a) If there is at most one type *T* in the set of subject types, then let the *identified candidate type* be *T*.

   b) If there is more than one type in the set of subject types, then let the identified candidate type be the type *T* such that there is no other type template *T2* for which the <schema name> of the schema that contains *T2* precedes in *DP* the <schema name> of the schema that contains *T*.

6) The implicit <schema name> is the <schema name> of the type *T*.

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

- 3 Subclauses moved to Part 4

10 Additional common elements

10.1 <interval qualifier>

Function

Specify the precision of an interval data type.

Format

```
<interval qualifier> ::=
        <start field> TO <end field>
      | <single datetime field>

<start field> ::=
      <non-second datetime field>
          [ <left paren> <interval leading field precision> <right paren> ]

<end field> ::=
        <non-second datetime field>
      | SECOND [ <left paren> <interval fractional seconds precision> <right paren> ]

<single datetime field> ::=
        <non-second datetime field>
            [ <left paren> <interval leading field precision> <right paren> ]
      | SECOND [ <left paren> <interval leading field precision>
            [ <comma> <interval fractional seconds precision> ] <right paren> ]

<datetime field> ::=
        <non-second datetime field>
      | SECOND

<non-second datetime field> ::= YEAR | MONTH | DAY | HOUR | MINUTE

<interval fractional seconds precision> ::= <unsigned integer>

<interval leading field precision> ::= <unsigned integer>
```

Syntax Rules

1) There is an ordering of significance of <datetime field>s. In order from most significant to least significant, the ordering is: YEAR, MONTH, DAY, HOUR, MINUTE, and SECOND. A <start field> or <single datetime field> with an <interval leading field precision> *i* is more significant than a <start field> or <single datetime field> with an <interval leading field precision> *j* if *i*>*j*. An <end field> or <single datetime field> with an <interval fractional seconds precision> *i* is more significant than an <end field> or <single datetime field> with an <interval fractional seconds precision> *j* if *i*>*j*.

2) If TO is specified, then:

   a) <start field> shall not be less significant than <end field>,

b) If <start field> specified MONTH, then <end field> shall specify MONTH, and

c) if <start field> specifies YEAR, then <end field> shall specify either YEAR or MONTH.

3) The maximum value of <interval leading field precision> is implementation-defined, but shall not be less than 2.

4) The maximum value of <interval fractional seconds precision> is implementation-defined, but shall not be less than 6.

5) An <interval leading field precision>, if specified, shall be greater than 0 and shall not be greater than the implementation-defined maximum. If <interval leading field precision> is not specified, then an <interval leading field precision> of 2 is implicit.

6) An <interval fractional seconds precision>, if specified, shall be greater than or equal to 0 and shall not be greater than the implementation-defined maximum. If SECOND is specified and <interval fractional seconds precision> is not specified, then an <interval fractional seconds precision> of 6 is implicit.

Access Rules

None.

General Rules

1) An item qualified by an <interval qualifier> contains the datetime fields identified by the <interval qualifier>.

Case:

a) If the <interval qualifier> specifies a <single datetime field>, then the <interval qualifier> identifies a single <datetime field>. Any reference to the *most significant* or *least significant* <datetime field> of the item refers to that <datetime field>.

b) Otherwise, the <interval qualifier> identifies those datetime fields from <start field> to <end field>, inclusive.

2) An <interval leading field precision> specifies

Case:

a) If the <datetime field> is SECOND, then the number of decimal digits of precision before the specified or implied decimal point of the seconds <datetime field>.

b) Otherwise, the number of decimal digits of precision of the first <datetime field>.

3) An <interval fractional seconds precision> specifies the number of decimal digits of precision following the specified or implied decimal point in the <datetime field> SECOND.

4) If <single datetime field> is not specified and <start field> and <end field> are the same <datetime field>, then the <interval qualifier> is equivalent to a <single datetime field> that is that <datetime field>.

5) The length in positions of an item of type interval is computed as follows.

Case:

a) If the item is a year-month interval, then

   Case:

   i) If the <interval qualifier> is a <single datetime field>, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field>.

   ii) Otherwise, the length in positions of the item is the implicit or explicit <interval leading field precision> of the <start field> plus 2 (the length of the <non-second datetime field> that is the <end field>) plus 1 (the length of the <minus sign> between the <years value> and the <months value> in a <year-month literal>).

b) Otherwise,

   Case:

   i) If the <interval qualifier> is a <single datetime field> that does not specify SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field>.

   ii) If the <interval qualifier> is a <single datetime field> that specifies SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision> of the <single datetime field> plus the implicit or explicit <interval fractional seconds precision>. If <interval fractional seconds precision> is greater than zero, then the length in positions of the item is increased by 1 (the length in positions of the <period> between the <seconds integer value> and the <seconds fraction>).

   iii) Otherwise, let *participating datetime fields* mean the datetime fields that are less significant than the <start field> and more significant than the <end field> of the <interval qualifier>. The length in positions of each participating datetime field is 2.

   Case:

   1) If <end field> is SECOND, then the length in positions of the item is the implicit or explicit <interval leading field precision>, plus 3 times the number of participating datetime fields (each participating datetime field has length 2 positions, plus the <minus sign>s or <colon>s that precede them have length 1 position), plus the implicit or explicit <interval fractional seconds precision>, plus 1 (the length in positions of the <colon> preceding the <end field>). If <interval fractional seconds precision> is greater than zero, then the length in positions of the item is increased by 1 (the length in positions of the <period> within the field identified by the <end field>).

   2) Otherwise, the length in positions of the item is the implicit or explicit <interval leading field precision>, plus 3 times the number of participating datetime fields (each participating datetime field has length 2 positions, plus the <minus sign>s or <colon>s that precede them have length 1 position), plus 2 (the length in positions of the <end field>), plus 1 (the length in positions of the <colon> preceding the <end field>).

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) If TO is specified, then:

      a) <start field> shall be more significant than <end field>,

      b) <start field> shall not specify MONTH, and

      c) if <start field> specifies YEAR, then <end field> shall specify MONTH.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <interval qualifier>.

10.2 <language clause>

Function

Specify a standard programming language.

Format

```
<language clause> ::=
      LANGUAGE <language name>

<language name> ::=
      ADA | C | COBOL | FORTRAN | MUMPS | PASCAL | PLI | SQL
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) The standard programming language specified by the clause is defined in the
   ANSI American
   ISO International
   Standard identified by the <language name> keyword. Table 17, "Standard programming languages", specifies the relationship.

Table 17—Standard programming languages

ANSI Only–SQL3

Language keyword	Relevant standard
ADA	ANSI/MIL-STD-1815A
C	ANSI X3.159
COBOL	ANSI X3.23
FORTRAN	ANSI X3.9 and ANSI X3.198
MUMPS	ANSI/MDC X11.1
PASCAL	ANSI/IEEE 770/X3.97 and ANSI/IEEE 770/X3.160
PLI	ANSI X3.53
SQL	ANSI X3.135-199*x*

ISO Only–SQL3

ADA	ISO/IEC 8652
C	ISO/IEC 9899
COBOL	ISO 1989
FORTRAN	ISO 1539
MUMPS	ISO/IEC 11756
PASCAL	ISO/IEC 7185 and ISO/IEC 10206
PLI	ISO 6160
SQL	ISO/IEC 9075:199*x*

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <language clause> shall not specify MUMPS.

- 1 Subclause moved to Part 4

10.3 <generated type reference>

Function

Specify a type generated from a type template.

Format

```
<generated type reference> ::=
       <type template name> <template parameter list>

<template parameter list> ::=
       <left paren>
         <template parameter> [ { <comma> <template parameter> }... ]
       <right paren>

<template parameter> ::=
         <value specification>
       | <data type>
```

Syntax Rules

1) A <template parameter> that is a <value specification> shall be a <literal> or a <template parameter name>.

2) Let *CTT* be the identified candidate <type template definition>. Let *GT* be an <abstract data type definition> derived from *CTT* as follows:

   a) The keyword TEMPLATE is removed.

   b) The <template parameter declaration list> is removed.

   c) Each occurrence of a <template parameter name> in the <abstract data type body> of *CTT* is replaced with the corresponding <template parameter> of the <generated type reference>.

   d) Two <generated type reference>s are said to be matching if

      i) the <type template name> and <template parameter list> of each identifies the same type template family, and

      ii) the number of <template parameter>s of each is the same, and

      iii) for each pair of corresponding parameters:

         1) they both identify the same <data type>, or

         2) the values of each are equal.

   e) The <type template name> is replaced by an implementation-dependent name *IDN* such that:

      i) a subsequent matching <generated type reference> yields the same implementation-dependent name *IDN*, and

ii) a subsequent non-matching <generated type reference> yields a different implementation-dependent name.

f) Each occurrence of :GEN_TYPE is replaced by *IDN*.

Note: "identified candidate <type template definition>" is defined in Subclause 5.4, "Names and identifiers".

3) *GT* shall conform to the Syntax Rules for <abstract data type definition>.

4) The <generated type reference> is effectively replaced by *IDN*.

Access Rules

1) The applicable privileges shall include USAGE for the type template family identified by <type template name>.

General Rules

1) If an <abstract data type definition> for *IDN* does not exist, then:

   a) The General Rules for the statement *GT* are executed.

   b) The abstract data type descriptor is augmented to identify the <type template name> of the <generated type reference>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <generated type reference>.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

10.4 <privileges>

Function

Specify privileges.

Format

```
<privileges> ::=
        ALL SCHEMA PRIVILEGES
      | <object privileges> ON <object name>

<object name> ::=
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
        [ <table type> ] <table name>
```

ISO Only--SQL3

```
        [ TABLE ] <table name>

      | DOMAIN <domain name>
      | COLLATION <collation name>
      | CHARACTER SET <character set name>
      | TRANSLATION <translation name>
      | NULL CLASS <null class name>
      | DATA TYPE <abstract data type name>
      | DATA TYPE <distinct type name>
      | MODULE <module name>
      | TYPE TEMPLATE <specific type template designator>
```

- 1 alternative moved to Part 4

```
<object privileges> ::=
        ALL PRIVILEGES
      | <action> [ { <comma> <action> }... ]

<action> ::=
        SELECT [ <left paren> <privilege column list> <right paren> ]
      | DELETE
      | INSERT [ <left paren> <privilege column list> <right paren> ]
      | UPDATE [ <left paren> <privilege column list> <right paren> ]
      | REFERENCES [ <left paren> <privilege column list> <right paren> ]
      | USAGE
      | TRIGGER
      | EXECUTE
      | UNDER

<privilege column list> ::= <column name list>

<grantee> ::=
        PUBLIC
      | <authorization identifier>
      | <role name>
```

Syntax Rules

1) If the <object name> of the <grant statement> or <revoke statement> specifying <privileges> specifies <table name>, then let *T* be the table identified by that <table name>. *T* shall not be a declared local temporary table.

ANSI Only—caused by ISO changes not yet considered by ANSI

If a <table type> contained in <object name> is not TABLE, then it shall be the same as that in the descriptor for the table identified by the <table name> contained in <object name>.

2) If <object name> specifies a <domain name>, <collation name>, <specific type template designator>, <character set name>, <translation name>, <distinct type name>, or <null class name>, then <privileges> shall specify USAGE. If <object name> specifies an <abstract data type name>, then <privileges> may specify USAGE. Otherwise, USAGE shall not be specified.

3) If <object name> specifies a <table name> that identifies a base table, then <privileges> may specify TRIGGER; otherwise, TRIGGER shall not be specified.

4) If <object name> specifies an <abstract data type name>, then <privileges> may specify UNDER; otherwise, UNDER shall not be specified.

5) If <object name> specifies MODULE, then <privileges> shall specify EXECUTE; otherwise, EXECUTE shall not be specified.

6) If *T* is a temporary table, then <privileges> shall specify ALL PRIVILEGES.

7) The <privileges> specify one or more privileges on the identified <schema element>s as follows:

   a) ALL SCHEMA PRIVILEGES specifies all privileges on all <schema element>s in the containing schema, including <schema element>s subsequently added by Schema Manipulation Language statements.

   b) The <object privileges> specify one or more privileges on the object identified by <object name>.

8) Each <column name> in a <privilege column list> shall identify a column of *T*.

9) UPDATE (<privilege column list>) is equivalent to the specification of UPDATE (<column name>) for each <column name> in <privilege column list>. INSERT (<privilege column list>) is equivalent to the specification of INSERT (<column name>) for each <column name> in <privilege column list>. REFERENCES (<privilege column list>) is equivalent to the specification of REFERENCES (<column name>) for each <column name> in <privilege column list>. SELECT (<privilege column list>) is equivalent to the specification of SELECT (<column name>) for each <column name> in <privilege column list>.

10) ALL PRIVILEGES is equivalent to the specification of all of the privileges on <object name> for which the current <authorization identifier> has grantable privilege descriptors.

- 1 Rule moved to Part 4

Access Rules

None.

General Rules

1) A <grantee> of PUBLIC denotes at all times a list of <grantee>s containing all of the <authorization identifier>s in the SQL environment.

2) For an <authorization identifier>, the user privileges includes those privileges defined by the privilege descriptors that define privileges whose grantee is either the <authorization identifier> or PUBLIC.

3) The set of applicable privileges for an <authorization identifier> includes those privileges defined by privilege descriptors associated with that <authorization identifier> or by privilege descriptors associated with the enabled applicable roles of that <authorization identifier>, together with those defined by privilege descriptors associated with PUBLIC.

4) SELECT (<column name>) specifies the SELECT privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then SELECT specifies the SELECT privilege on all columns of *T* including any columns subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

5) UPDATE (<column name>) specifies the UPDATE privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then UPDATE specifies the UPDATE privilege on all columns of *T*, including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

6) INSERT (<column name>) specifies the INSERT privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then INSERT specifies the INSERT privilege on all columns of *T*, including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

7) REFERENCES (<column name>) specifies the REFERENCES privilege on the indicated column and implies one or more column privilege descriptors. If the <privilege column list> is omitted, then REFERENCES specifies the REFERENCES privilege on all columns of *T*, including any column subsequently added to *T* and implies a table privilege descriptor and one or more column privilege descriptors.

8) A role *A* identified by <role name> is said to contain the set of roles identified by role authorization descriptors as having been granted to *A*, together with all other roles that are contained by roles in the set.

9) The set of applicable roles for an <authorization identifier> *B* includes all roles defined by role authorization descriptors as having been granted to *B* or to PUBLIC, together with all other roles that are contained in roles already in the set.

10) The set of enabled applicable roles for an <authorization identifier> *B* includes the role *R* enabled by the most recent <set role statement> together with all roles contained in *R*.

11) *B* has the WITH
ANSI GRANT
ISO ADMIN
OPTION on a role if a role authorization descriptor identifies the role as granted to *B* with the WITH
ANSI GRANT
ISO ADMIN
OPTION or a role authorization descriptor identifies it as granted with the WITH
ANSI GRANT
ISO ADMIN
OPTION to another applicable role for *B*.

****Editor's Note****
The absence of any General Rules specifying the meaning and semantics of the UNDER privilege has been identified as a Possible Problem. See Possible Problem **322** in the Editor's Notes.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <action> shall not specify EXECUTE, TRIGGER, or UNDER.

   b) An <action> that specifies SELECT shall not contain a <privilege column list>.

   c) An <object name> shall not specify DATA TYPE or TYPE TEMPLATE.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) An <action> that specifies INSERT shall not contain a <privilege column list>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

10.5 <character set specification>

Function

Identify a character set.

Format

```
<character set specification> ::=
         <standard character repertoire name>
       | <implementation-defined character repertoire name>
       | <user-defined character repertoire name>
       | <standard universal character form-of-use name>
       | <implementation-defined universal character form-of-use name>

<standard character repertoire name> ::= <character set name>

<implementation-defined character repertoire name> ::= <character set name>

<user-defined character repertoire name> ::= <character set name>

<standard universal character form-of-use name> ::=
       <character set name>

<implementation-defined universal character form-of-use name> ::=
       <character set name>
```

Syntax Rules

1) The <standard character repertoire name>s, <implementation-defined character repertoire name>s, <standard universal character form-of-use name>s, and <implementation-defined universal character form-of-use name>s that are supported are implementation-defined.

2) A character set identified by a <standard character repertoire name>, by an <implementation-defined character repertoire name>, by a <standard universal character form-of-use name>, or by an <implementation-defined universal character form-of-use name> has associated with it a privilege descriptor that was effectively defined by the <grant statement>

   GRANT USAGE ON CHARACTER SET *CS* TO PUBLIC WITH GRANT OPTION

   where *CS* is the <character set name> contained in the <character set specification>. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".

3) The <implementation-defined character repertoire name>s shall include SQL_TEXT.

4) Let *C* be the <character set name> contained in the <character set specification>. If the <character set specification> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <character set name> shall include the descriptor of *C*. If the <character set specification> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <character set name> should include the descriptor of *C* or *S* shall include a <schema element> that creates the descriptor of *C*.

Access Rules

1) The applicable privileges shall include USAGE on *C*.

General Rules

1) A <character set specification> identifies a character set. Let the identified character set be *CS*.

   Note: A character set comprises the characters in the character set's repertoire together with a form-of-use that specifies the convention for arranging those characters into character strings.

2) A <standard character repertoire name> specifies the name of a character repertoire that is defined by a national or international standard. The character repertoire and form-of-use of *CS*, implied by the <standard character repertoire name>, are defined by the standard that defined that <standard character repertoire name>. The default collating sequence of the character repertoire is defined by the order of the characters in the standard and has the PAD SPACE attribute.

3) An <implementation-defined character repertoire name> specifies the name of a character repertoire that is implementation-defined. The character repertoire and form-of-use of *CS*, implied by the <implementation-defined character repertoire name>, are implementation-defined. The default collating sequence of the character repertoire and whether the collating sequence has the NO PAD attribute or the PAD SPACE attribute implementation-defined.

4) A <user-defined character repertoire name> identifies a character set whose descriptor is in some schema whose <schema name> is not INFORMATION_SCHEMA.

   Note: The default collating sequence of the character repertoire is defined as in Subclause 11.37, "<character set definition>".

5) A <standard universal character form-of-use name> identifies form-of-use that is defined by some national or international standard. That form-of-use is the form-of-use of *CS*. The character repertoire of *CS* is as defined in that standard. The default collating sequence of the character repertoire is defined by the order of the characters in ISO/IEC DIS 10646 and has the PAD SPACE attribute.

   Note: Specific forms-of-use implied by this rule include ISO 2022 code extension techniques.

6) An <implementation-defined universal character form-of-use name> identifies an implementation-defined form-of-use that shall be the form-of-use of *CS*. The implied character repertoire and default collating sequence of *CS* and whether the collating sequence has the NO PAD attribute or the PAD SPACE attribute are implementation-defined.

   Note: Specific forms-of-use implied by this rule include implementation-defined techniques such as mixed one-octet/two-octet Latin/Kanji or Compound String.

7) There is a character set descriptor for every character set that can be specified by a <character set specification>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <character set specification>.

10.6 <collate clause>

Function

Specify a default collating sequence.

Format

```
<collate clause> ::= COLLATE <collation name>
```

Syntax Rules

1) Let *C* be the <collation name> contained in the <collate clause>. If the <collate clause> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <collation name> shall include the descriptor of *C*. If the <collate clause> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <collation name> shall include the descriptor of *C* or *S* shall include a <schema element> that creates the descriptor of *C*.

Access Rules

1) The applicable privileges shall include USAGE on *C*.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

10.7 <constraint name definition> and <constraint attributes>

Function

Specify the name of a constraint and its attributes.

Format

```
<constraint name definition> ::= CONSTRAINT <constraint name>

<constraint attributes> ::=
         <constraint check time> [ [ NOT ] DEFERRABLE ]
       | [ NOT ] DEFERRABLE [ <constraint check time> ]

<constraint check time> ::=   INITIALLY DEFERRED | INITIALLY IMMEDIATE
```

Syntax Rules

1) If a <constraint name definition> is contained in a <schema definition>, and if the <constraint name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

2) The <qualified identifier> of <constraint name> shall be different from the <qualified identifier> of the <constraint name> of any other constraint defined in the same schema.

3) If <constraint check time> is not specified, then INITIALLY IMMEDIATE is implicit.

4) Case:

   a) If INITIALLY DEFERRED is specified, then:

      i) NOT DEFERRABLE shall not be specified.

      ii) If DEFERRABLE is not specified, then DEFERRABLE is implicit.

   b) If INITIALLY IMMEDIATE is specified or implicit and neither DEFERRABLE nor NOT DEFERRABLE is specified, then NOT DEFERRABLE is implicit.

Access Rules

None.

General Rules

1) If NOT DEFERRABLE is specified, then the constraint is not deferrable; otherwise it is deferrable.

2) If <constraint check time> is INITIALLY DEFERRED, then the initial constraint mode for the constraint is *deferred*; otherwise, the initial constraint mode for the constraint is *immediate*.

3) If, on completion of any SQL-statement, the constraint mode of any constraint is immediate, then that constraint is effectively checked.

   Note: This includes the cases where SQL-statement is a <set constraints mode statement>, a <commit statement>, or the statement that causes a constraint with a constraint mode of *initially immediate* to be created.

4) When a constraint is effectively checked, if the constraint is not satisfied, then an exception condition is raised: *integrity constraint violation*. If this exception condition is raised as a result of executing a <commit statement>, then SQLSTATE is not set to *integrity constraint violation*, but is set to *transaction rollback—integrity constraint violation* (see the General Rules of Subclause 14.7, "<commit statement>").

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no explicit <constraint attributes>.
      Note: This means that INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall contain no <constraint name definition>.

• 1 Subclause moved to Part 4

10.8 <specific type template designator>

Function

Specify a type template.

Format

```
<specific type template designator> ::=
      <type template name> [ <template parameter type list> ]

<template parameter type list> ::=
      <left paren> <template parameter type> [ { <comma> <template parameter type> }... ] <right pare
```

Syntax Rules

1) Let *TN* be the <type template name>. Let *TF* be the type template family of *TN*.

2) Case:

   a) If a <template parameter type list> is specified, then for all *i* there shall be exactly one member type template in *TF* whose *i*-th <template parameter declaration> specifies a <template parameter type> that is identical to the *i*-th <template parameter type> in this <template parameter type list>. The <specified type template designator> identifies that type template.

   b) Otherwise, there shall be exactly one member in *TF*. The <specific type template designator> identifies that type template.

Access Rules

None.

General Rules

None.

11 Schema definition and manipulation

11.1 <schema definition>

Function

Define a schema.

Format

```
<schema definition> ::=
       CREATE SCHEMA <schema name clause>
         [ <schema character set specification> ]
         [ <schema path specification> ]
         [ <schema element>... ]



<schema name clause> ::=
         <schema name>
       | AUTHORIZATION <schema authorization identifier>
       | <schema name> AUTHORIZATION <schema authorization identifier>

<schema authorization identifier> ::=
       <authorization identifier>

<schema character set specification> ::=
       DEFAULT CHARACTER SET <character set specification>

<schema path specification> ::=
       PATH <schema name list>

<schema name list> ::=
       <schema name> [ { <comma> <schema name> }... ]

<schema element> ::=
         <table definition>
       | <view definition>
       | <domain definition>
       | <null class definition>
       | <character set definition>
       | <collation definition>
       | <translation definition>
       | <assertion definition>
       | <trigger definition>
```

- 1 alternative moved to Part 4

```
       | <abstract data type definition>
       | <distinct type definition>
       | <type template definition>
       | <operators definition>
       | <grant statement>
       | <role definition>
       | <grant role statement>
```

Syntax Rules

1) If <schema name> is not specified, then a <schema name> equal to <schema authorization identifier> is implicit.

2) If AUTHORIZATION <schema authorization identifier> is not specified, then

   Case:

   a) If the <schema definition> is contained in a <module> that has a <module authorization identifier> specified, then an <authorization identifier> equal to that <module authorization identifier> is implicit for the <schema definition>.

   b) Otherwise, an <authorization identifier> equal to the SQL-session <authorization identifier> is implicit.

3) The <unqualified schema name> of the explicit or implicit <schema name> shall be different from the <unqualified schema name> of the <schema name> of any other schema in the catalog identified by the <catalog name> of <schema name>.

4) If a <schema definition> appears in a <routine> in a <module>, then the effective <schema authorization identifier> and <schema name> during processing of the <schema definition> is the <schema authorization identifier> and <schema name> specified or implicit in the <schema definition>. Other SQL-statements executed in <routine>s in the <module> have the <module authorization identifier> and <schema name> specified or implicit for the <module>.

   Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
   ANSI American
   ISO International
   Standard.

5) If <schema character set specification> is not specified, then a <schema character set specification> containing an implementation-defined <character set specification> is implicit.

6) If <schema path specification> is not specified, then a <schema path specification> containing an implementation-defined <schema name list> that includes the <schema name> contained in <schema name clause> is implicit.

ISO Only–SQL3

7) If <schema name list> is contained in a <schema definition>, then the explicit or implicit <catalog name> of each <schema name> contained in <schema name list> shall be the same as the <catalog name> of the <schema name> contained in the <schema name clause>.

ANSI Only—caused by ISO changes not yet considered by ANSI

8) Each <schema name> contained in <schema name list> shall conform to the Format and Syntax Rules of <schema name>.

9) The explicit or implicit <catalog name> of each <schema name> contained in <schema name list> shall be the same as the <catalog name> of the <schema name> contained in <schema name clause>.

---

10) No <schema name> in <schema name list> shall be specified more than once.

11) A <schema definition> shall contain at most one <operators definition>.

Access Rules

1) The privileges necessary to execute the <schema definition> are implementation-defined.

General Rules

1) A schema *S* is created with a name equal to the explicit or implicit <schema name> and a default character set name equal to the <character set specification> of the explicit or implicit <default character set specification>.

2) The <schema authorization identifier> is the current <authorization identifier> for privilege determination for *S*.

3) Those objects defined by <schema element>s and their associated descriptors are effectively created.

4) The explicit or implicit <character set specification> is used as the default character set used for all <column definition>s and <domain definition>s that do not specify an explicit character set.

5) The explicit or implicit <schema name list> is used as the default SQL-path of the schema. The default SQL-path is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in the <schema definition>.

Note: <routine name> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <null class definition>.

   b) Conforming Full SQL language shall not contain any <trigger definition>.

   c) Conforming Full SQL language shall not contain any <external function declaration>.

• 1 Rule moved to Part 4

   d) Conforming Full SQL language shall not contain any <abstract data type definition>.

   e) Conforming Full SQL language shall not contain any <distinct type definition>.

   f) Conforming Full SQL language shall not contain any <type template definition>.

   g) Conforming Full SQL language shall not contain any <role definition>.

h) Conforming Full SQL language shall not contain any <grant role statement>.

i) Conforming Full SQL language shall not contain any <schema path specification>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

b) Conforming Intermediate SQL language shall not contain any <collation definition>.

c) Conforming Intermediate SQL language shall not contain any <translation definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <domain definition>.

b) A <schema name clause> shall specify AUTHORIZATION and shall not specify a <schema name>.

c) A <schema character set specification> shall not be specified.

d) Conforming Entry SQL language shall not contain any <character set definition>.

11.2 <alter schema statement>

Function

Change the definition of a schema.

Format

```
<alter schema statement> ::=
       ALTER SCHEMA <schema name>
         <alter schema action>

<alter schema action> ::=
         <add operators definition>
```

Syntax Rules

1) The <schema name> shall identify a schema.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name>.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain an <alter schema statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.3 <add operators definition>

Function

Add new operator definitions to a schema.

Format

```
<add operators definition> ::=
      ADD <operators definition body>
```

Syntax Rules

1) Let *S* be the schema identified by the <schema name> of the containing <alter schema statement>.

2) Let *NEWOD* be the <operators definition body>. If *S* contains an <operators definition>, then let *OLDOD* be that <operators definition>; otherwise, let *OLDOD* be empty.

3) For *k* ranging from 10 to 19, and for *FORM* as INFIX, PREFIX, and POSTFIX, if *OLDID* contains an <operator group> *OLDOPG* whose <operator level> is *k* and whose <operator form> is *FORM*, then:

   Case:

   a) If *NEWOD* contains an <operator group> *NEWOPG* whose <operator level> is *k* and whose <operator form> is *FORM*, then insert each <operator> contained in *OLDOPG* into *NEWOPG*.

   b) Otherwise, insert *OLDOPG* into *NEWOD*.

4) *NEWOD* shall be a valid <operators definition>.

Access Rules

None.

General Rules

1) *NEWOD* becomes the <operators definition> of *S*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain an <add operators definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.4 <drop schema statement>

Function

Destroy a schema.

Format

```
<drop schema statement> ::=
       DROP SCHEMA <schema name> <drop behavior>

<drop behavior> ::= CASCADE | RESTRICT
```

Syntax Rules

1) Let *S* be the schema identified by <schema name>.

2) *S* shall identify a schema in the catalog identified by the explicit or implicit <catalog name>.

3) If RESTRICT is specified, then *S* shall not contain any persistent base tables, global temporary tables, created local temporary tables, views, domains, assertions, external functions, SQL functions, triggers, modules, abstract data types, distinct types, type template definitions, character sets, collations, or translations.

   Note: If CASCADE is specified, then such objects will be dropped by the effective execution of the SQL schema manipulation statements specified in the General Rules of this Subclause.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name>.

General Rules

1) Let *T* be the <table name> of any base table or temporary table contained in *S*. The following <drop table statement> is effectively executed:

   DROP TABLE *T* CASCADE

2) Let *V* be the <table name> of any view or temporary view contained in *S*. The following <drop view statement> is effectively executed:

   DROP VIEW *V* CASCADE

3) Let *D* be the <domain name> of any domain contained in *S*. The following <drop domain statement> is effectively executed:

   DROP DOMAIN *D* CASCADE

4) Let *A* be the <constraint name> of any assertion contained in *S*. The following <drop assertion statement> is effectively executed:

   DROP ASSERTION *A*

5) Let *CD* be the <collation name> of any collating sequence contained in *S*. The following <drop collation statement> is effectively executed:

   DROP COLLATION *CD* CASCADE

6) Let *TD* be the <translation name> of any translation contained in *S*. The following <drop translation statement> is effectively executed:

   DROP TRANSLATION *TD*

7) Let *RD* be the <character set name> of any character set contained in *S*. The following <drop character set statement> is effectively executed:

   DROP CHARACTER SET *RD*

8) Let *EF* be the <external function name> of any external function contained in *S*. The following <drop external function statement> is effectively executed:

   DROP EXTERNAL FUNCTION *EF* CASCADE

****Editor's Note****

The "<drop external function statement>" in the previous Rule is a reference to an undefined production. Have I missed something?

9) Let *SF* be the <SQL function name> of any SQL function contained in *S*. The following <drop routine statement> is effectively executed:

   DROP SQL FUNCTION *SF* CASCADE

****Editor's Note****

The preceding statement does not appear to be a "<drop routine statement>" as advertised. Have I missed something?

Chris Farrar has pointed out that "Correcting this problem is not fully trivial, as any correction must deal with dropping routines that are members of routine families."

Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

10) Let *DT* be the <abstract data type name> of any abstract data type contained in *S* or the <distinct type name> of any distinct type contained in *S*. The following <drop data type statement> is effectively executed:

   DROP DATA TYPE *DT* CASCADE

11) Let *TDT* be the <type template name> of any type template contained in *S*. The following <drop type template statement> is effectively executed:

   DROP TYPE TEMPLATE *TDT* CASCADE

12) Let *TT* be the <trigger name> of any trigger contained in *S*. The following <drop trigger statement> is effectively executed:

   DROP TRIGGER *TT*

13) Let *N* be the <null class name> of any null class contained in *S*. The following <drop null class statement> is effectively executed:

    DROP NULL CLASS *N*

14) The identified schema and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

    None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

    None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

    a) Conforming Entry SQL language shall not contain a <drop schema statement>.

11.5 <table definition>

Function

Define a persistent base table, a created local temporary table, or a global temporary table.

ANSI Only—caused by ISO changes not yet considered by ANSI

A table may be list, set, or multiset table.

Format

```
<table definition> ::=
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
      CREATE [ <table scope> ] [ <table type> ] TABLE <table name>
```

ISO Only--SQL3

```
      CREATE [ <table scope> ] TABLE <table name>
```

```
        [ <constant or updatable> ]
        { <table element list> | <subtable clause> }
        [ ON COMMIT <table commit action> ROWS ]
```

ANSI Only--SQL3

```
      [ WITH IDENTITY ]
```

```
<table scope> ::=
      <global or local> TEMPORARY

<global or local> ::=
        GLOBAL
      | LOCAL
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
<table type> ::=
        MULTISET
      | SET
      | LIST
```

```
<table commit action> ::=
        PRESERVE
      | DELETE
```

```
<constant or updatable> ::=
         CONSTANT
       | UPDATABLE

<subtable clause> ::=
       UNDER <supertable clause> [ { , <supertable clause> }... ]

<supertable clause> ::=
       <supertable name>
         [ WITH ( <member renaming element> [ { , <member renaming element> }... ] ) ]

<member renaming element> ::=
       <supertable member name> AS <subtable member name>

<supertable member name> ::= <column name>

<subtable member name> ::= <column name>

<supertable name> ::= <table name>

<table element list> ::=
       <left paren> <table element> [ { <comma> <table element> }... ] <right paren>

<table element> ::=
         <column definition>
       | <table constraint definition>
       | <like clause>

<like clause> ::= LIKE <table name>
```

Syntax Rules

1) If a <table definition> is contained in a <schema definition> *SD* and the <table name> contains a <local or schema qualifier>, then that <local or schema qualifier> shall be the same as the implicit or explicit <schema name> of *SD*.

2) The schema identified by the explicit or implicit schema name of the <table name> shall not include a table descriptor whose table name is <table name>.

3) If the <table definition> appears in a <schema definition>, then let *A* be the explicit or implicit <authorization identifier> of the <schema definition>. Otherwise, let *A* be the <authorization identifier> of the current SQL-session.

ANSI Only—caused by ISO changes not yet considered by ANSI

4) If a <table type> of SET is specified, then the following <table constraint definition> is implicit as a <table element> in the <table element list>:

   UNIQUE (VALUE)

   ANSI Only–SQL3

5) No <column definition> shall specify a <column name> of IDENTITY.

6) If *SD* defines a base table, then

Case:

a) If WITH IDENTITY is specified, then a row identifier is explicitly defined for the table.

b) Otherwise, it is implementation-defined whether a row identifier for the table is implicitly defined.

---

7) If LIKE is specified, then:

a) Let *T1* be the table identified in the <like clause>.

b) The <like clause> is effectively replaced by *columns* where *columns* contains a <column definition> for every column that is defined in *T1*, in the order in which the columns appear in *T1*.

Note: <column constraint>s are not included in *columns*; <column constraint>s are effectively transformed to <table constraints>s and are thereby excluded.

ANSI Only–SQL3

---

8) If a <table definition> or <temporary table declaration> *TTD* contains a <subtable clause>, then:

Case:

a) If *TTD* is a persistent base table, then each <table name> in a <supertable clause> shall identify a persistent base table.

b) If *TTD* is a global temporary table, then each <table name> in a <supertable clause> shall identify a global temporary table.

c) If *TTD* is a created local temporary table, then each <table name> in a <supertable clause> shall identify either a global temporary table or a created local temporary table.

d) If *TTD* is a declared local temporary table, then each <table name> in a <supertable clause> shall identify either a global temporary table, a created local temporary table or a declared local temporary table.

---

9) Case:

a) If CONSTANT is specified, then CONSTANT is implicit for each <stored column> contained in the <table element list>.

b) If UPDATABLE is specified, then there shall exist at least one <stored column> contained in the <table element list> that does not specify CONSTANT.

10) If ON COMMIT is specified, then TEMPORARY shall be specified.

11) If TEMPORARY is specified and ON COMMIT is not specified, then ON COMMIT DELETE ROWS is implicit.

12) A <table element list> shall contain at least one <column definition>.

ISO Only—caused by ANSI changes not yet considered by ISO

---

13) No <column name> immediately contained in a <column definition> explicitly specified in the <table definition>, or contained in a <member renaming element>, shall specify OID.

****Editor's Note****

The author of X3H2-92-062rev1/OTT-008 adjusted the position of a number of Syntax Rules, both in this Subclause and in other Subclauses, saying that the moved Rule should precede or follow other Syntax Rules that have "implicit" actions. This appears to be fallacious based on our statement in Concepts that the Syntax and Access Rules are effectively applied simultaneously! Is there a problem here?

---

14) The scope of the <table name> is the <table definition>.

15) Let *T* be the table defined by the <table definition> *TD*.

16) If *TD* specifies a <subtable clause>, then:

    a) Each <supertable name> identifies a direct supertable of *T*.

    b) *T* is a direct subtable of each table identified by a <supertable name> in <subtable clause>.

17) Let the term *candidate inherited column* refer to a column whose column descriptor is defined within the table descriptor of a direct supertable of *T*.

18) Let the term *replicated column* mean a column appearing in more than one direct supertable of *T* that is inherited by at least one of those direct supertables from the same column of a single higher-level supertable. The column name of a replicated column need not be the same in all the tables in which it appears.

19) The inherited columns of *T* and their ordinal positions are determined as follows:

    a) Let *S* be the list of candidate inherited columns, constructed as follows:

    $$C_{11}, \cdots, C_{1k(1)}, \cdots, C_{i1}, \cdots, C_{ik(i)}, \cdots, C_{m1}, \cdots, C_{mk(m)}$$

    where *m* is the number of direct supertables of *T* and *k*(*i*) is the degree of the *i*-th supertable T_i of *T*. The list *S* preserves the order of the supertables T_i of *T* and the order of the columns in each T_i.

    b) If *SC* is a replicated column, then all occurrences of *SC* except the first are dropped from *S*.

    c) If the *j*-th column in *S* is the subject of a <member renaming element>, then the column name of the *j*-th column of *S* is specified by the <member renaming clause>.

d) The inherited columns and their ordinal positions are the columns of *S* and their ordinal positions in *S*.

Note: The ordinal position of columns subsequently added to the table by means of ALTER TABLE is determined by the rules of Subclause 11.6, "<column definition>".

e) No column name of a column in *S* shall appear more than once.

20) The phrase *originally-defined column* refers to a column of *T* that is not an inherited column.

21) If a <table definition> *TD* specifies a <subtable clause>, then:

a) The table *T* defined by *TD* shall not have *T* as a supertable.

b) The descriptor of the schema identified by the explicit or implicit <schema name> of <table name> shall include the descriptor of the table identified by <supertable name>.

c) No <supertable name> shall appear more than once in the <subtable clause> of *TD*.

ANSI Only–SQL3

d) A row identifier is implicitly defined for *T*. For every table *ST* named in the <subtable clause>, a row identifier is implicitly defined for *ST*.

ISO Only—caused by ANSI changes not yet considered by ISO

e) The <table definition> of any table identified by a <supertable name> in the <subtable clause> of *TD* shall contain an explicit or implicit <unique specification> that specifies PRIMARY KEY.

f) Each <supertable member name> in a <supertable clause> shall identify a column of the supertable identified by the <supertable name> in that <supertable clause>, and the same column shall not be identified more than once.

g) There shall be exactly one maximal supertable in the subtable family to which the supertable belongs.

Access Rules

1) If a <table definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <table name>.

2) If a <like clause> is contained in a <table definition>, then *A* shall have SELECT privilege on the table identified in that <like clause>.

General Rules

1) A <table definition> defines either a persistent base table, a global temporary table or a created local temporary table. If GLOBAL is specified, then a global temporary table is defined. If LOCAL is specified, then a created local temporary table is defined. Otherwise, a persistent base table is defined.

2) The degree of *T* is intially set to *NIC*, the number of inherited columns; the General Rules of Subclause 11.6, "<column definition>", specify the degree of *T* during the definition of the originally-defined columns of *T*.

ANSI Only–SQL3

---

3) If a row identifier is defined for *T*, then a new subtype *RDT* of the row identifier type is created. An implicit column of type *RDT* and name IDENTITY is created in *T*. *RDT* is a subtype of the row identifier defined for every supertable of *T*.

---

4) A table descriptor *TDS* is created that describes *T*. *TDS* includes:

   a) The table name *TN*,

   b) The degree of *T*,

   ANSI Only—caused by ISO changes not yet considered by ANSI

   ---

   c) Case:

      i) If a <table type> of SET is specified, then the table descriptor contains an indication that this is a set table.

      ii) If a <table type> of LIST is specified, then the table descriptor contains an indication that this is a list table.

      iii) Otherwise, the table descriptor contains an indication that this is a multiset table.

      ---

   d) The descriptors of all inherited columns, their ordinal positions in *T*, and any <subtable member name>s for the columns contained in the <table definition>,

   e) The table names of each direct supertable of *T*, and

   f) The table constraint descriptors specified by each <table constraint definition>.

****Editor's Note****

The absence of GRs that explicitly say that the table is created when the <table definition> is executed as an <SQL procedure statement> in a <routine> has been identified as a Possible Problem. This applies to other Subclauses as well. See Possible Problem **205** in the Editor's Notes.

5) A set of privilege descriptors is created that define the privileges INSERT, SELECT, UPDATE, DELETE, and REFERENCES on this table and SELECT, INSERT, UPDATE, and REFERENCES for every <column definition> in the table definition to the <authorization identifier> of the <schema definition> or <module> in which the <table definition> appears. These privileges are grantable.

   The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM".

6) If a <table definition> *TD* specifies a <subtable clause>, then:

   a) The <table name> *T* of *TD* defines a subtable, called a *direct subtable*. Each <supertable name> specified in the <subtable clause> defines a supertable, called a *direct supertable*.

   b) Every <table element> of every supertable is inherited by the subtable *T*.

ANSI Only—caused by ISO changes not yet considered by ANSI

7) Case:

   a) If a <table type> of SET is specified, then the table being created is a set table.

   b) If a <table type> of LIST is specified, then the table being created is a list table.

   c) Otherwise, the table being created is a multiset table.

8) The row type of the table *T* defined by the <table definition> is the set of pairs (<field name>, <data type>) where <field name> is the name of a column *C* of *T* and <data type> is the data type of *C*. This set of pairs contain one pair for each column of *T*, in the order of their ordinal position in *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

ANSI Only—caused by ISO changes not yet considered by ANSI

   a) A <table definition> shall not specify a <table type>.

ANSI Only–SQL3

   b) A <table definition> shall not specify WITH IDENTITY.

   c) A <table definition> shall not specify a <constant or updatable>.

d) A <table definition> shall not specify a <subtable clause>.

e) A <table element> shall not be a <like clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not specify TEMPORARY and shall not reference any global or local temporary table.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.6 <column definition>

Function

Define a column of a table.

Format

```
<column definition> ::=
       <column name>
       { <data type> | <domain name> }
       [ <default clause> ]
       [ <column constraint definition>... ]
       [ <collate clause> ]
       [ <null clause> ]

<column constraint definition> ::=
       [ <constraint name definition> ]
       <column constraint> [ <constraint attributes> ]

<column constraint> ::=
         NOT NULL
       | <unique specification>
       | <references specification>
       | <check constraint definition>

<null clause> ::=
       NULL IS <null class name>
```

Syntax Rules

1) Case:

   a) If the <column definition> is contained in a <table definition>, then let *T* be the table defined by that <table definition>.

   b) If the <column definition> is contained in a <temporary table declaration>, then let *T* be the table declared by that <temporary table declaration>.

   c) If the <column definition> is contained in an <alter table statement>, then let *T* be the table identified in the containing <alter table statement>.

   The <column name> in the <column definition> shall be different from the <column name> of any other column of *T*.

2) Let *C* be the <column name> of the <column definition>.

3) If <domain name> is specified, then let *D* be the domain identified by the <domain name>.

4) The data type of the column is

   Case:

   a) If <data type> is specified, then that data type.

   b) Otherwise, the data type of *D*.

5) If the data type of the column is character string, then the collation of the column is

   Case:

   a) If <collate clause> is specified, then the collation specified by that <collate clause>.

   b) If <domain name> is specified, then the collation of *D*, if any.

   c) Otherwise, the default collation of the character set of the column.

   Note: The character set of a column is determined by its data type.

6) If a <data type> is specified, then:

   a) Let *DT* be the <data type>.

   b) If DT is CHARACTER, CHARACTER VARYING, or CHARACTER LARGE OBJECT and does not specify a <character set specification>, then the <character set specification> specified or implicit in the <schema character set specification> of the <schema definition> that created the schema identified by the <schema name> immediately contained in the <table name> of the containing <table definition> or <alter table statement> is implicit.

   c) If *DT* is a <character string type> that identifies a character set that specifies a <collate clause> and the <column definition> does not contain a <collate clause>, then the <collate clause> of the <character string type> is implicit in the <column definition>.

7) If <collate clause> is specified, then data type shall be a character string type.

8) If a <column constraint definition> is specified, then let *CND* be the <constraint name definition> if one is specified and let *CND* be a zero-length string otherwise; let *CA* be the <constraint attributes> if specified and let *CA* be a zero-length string otherwise. The <column constraint definition> is equivalent to a <table constraint definition> as follows:

   Case:

   a) If a <column constraint definition> is specified that contains the <column constraint> NOT NULL, then it is equivalent to a <table constraint definition> that contains the following <table constraint>:

      CND CHECK (*C* IS NOT NULL) *CA*

   b) If a <column constraint definition> is specified that contains a <unique specification>, then it is equivalent to a <table constraint definition> that contains the following <table constraint>:

      CND <unique specification> (*C*) *CA*

      Note: The <unique specification> is defined in Subclause 11.11, "<unique constraint definition>".

   c) If a <column constraint definition> is specified that contains a <references specification>, then it is equivalent to a <table constraint definition> that contains the following <table constraint>:

      CND FOREIGN KEY (*C*) <references specification> *CA*

      Note: The <references specification> is defined in Subclause 11.12, "<referential constraint definition>".

d) If a <column constraint definition> is specified that contains a <check constraint definition>, then it is equivalent to a <table constraint definition> that contains the following <table constraint>:

CND CHECK (<search condition>) *CA*

Each column reference directly contained in the <search condition> shall reference column *C*.

9) If <domain name> is specified or implied, then <null clause> shall not be specified.

10) If the <column definition> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <domain name> shall include the descriptor of *D*. If the <column definition> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <domain name> shall include the descriptor of *D* or *S* shall include a <schema element> that creates the descriptor of *D*.

ANSI Only–SQL3

11) If a <row identifier type> is specified without a <references specification>, then let *TN* be the <table name> specified in <row identifier type>; REFERENCES *TN* IDENTITY is implicit.

Access Rules

1) If <domain name> is specified, then the applicable privileges shall include USAGE on *D*.

2) If a <data type> is specified that is an abstract data type *A*, then the applicable privileges shall include USAGE on *A*.

3) If <null class name> is specified, then the applicable privileges shall include USAGE.

4) If a <data type> is specified that is a distinct type *DT*, then the applicable privileges shall include USAGE on *DT*.

General Rules

1) A <column definition> defines a column in a table.

2) The <collate clause> specifies the default collating sequence for the column. If <collate clause> is not specified, then the default collating sequence is that used for comparisons of *Coercible* coercibility attribute, as defined in Subclause 8.2, "<comparison predicate>".

3) If the <column definition> specifies <data type>, then a data type descriptor is created that describes the data type of the column being defined.

4) The degree of the table *T* being defined in the containing <table definition> or <temporary table declaration>, or being altered by the containing <alter table statement> is increased by 1.

5) A column descriptor is created that describes the column being defined. The column descriptor contains:

a) <column name>, the name of the column.

b) Case:

i) If the <column definition> specifies a <data type>, then the data type descriptor of the <data type> of the column.

ii) Otherwise, the domain of the column.

c) The ordinal position of the column, which is equal to the degree of *T*.

d) The nullability characteristic of the column, determined according to the rules in Subclause 4.19, "Columns, fields, and attributes", and, if applicable, the null class of the column.

e) If <default clause> is specified, then the <default option>.

f) If the <column definition> contains a <collate clause>, then the collation name.

6) Case:

a) If <domain name> is specified or implied, then the null class of this column is the null class of the domain specified by <domain name>.

b) If <null clause> is specified, then the null class of this column is the specified <null class name>.

c) Otherwise, the null class of this column is the general null class.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <column definition> shall not contain a <null clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <column definition> shall not contain a <domain name>.

b) A <column constraint> shall not contain a <referential triggered action>.

c) Conforming Entry SQL language shall not contain any <constraint name definition>.

11.7 <attribute definition>

Function

Define an attribute of an abstract data type.

Format

```
<attribute definition> ::=
       [ <encapsulation level> ]
       <attribute name> [ <constant or updatable> ]
       { <data type> | <domain name> }
```

- 1 option deleted.

```
       [ <column constraint definition>... ]
       [ <collate clause> ]
       [ <null clause> ]
```

- 5 productions deleted

```
<encapsulation level> ::=
         PRIVATE
       | PROTECTED
       | PUBLIC
```

Syntax Rules

1) If <attribute definition> is contained in an <abstract data type body> *ADTD*, then let *EL* be the <table element list> or <member list> of *ADTD*. If <encapsulation level> is not specified, then:

   Case:

   a) If this <attribute definition> is the first element of the list *EL* that contains either a <column definition> or a <routine declaration>, then <encapsulation level> PUBLIC is implicit.

   b) Otherwise, the <encapsulation level> of the nearest preceding element of the list *EL* that contains either an <attribute definition> or a <routine declaration> is implicit.

- 1 Subrule deleted.

2) If the <attribute definition> is contained in an <abstract data type definition>, then let *ADT* be the abstract data type identified in that <abstract data type definition>. The <attribute name> in the <attribute definition> shall be different from the <attribute name> of any other attribute of *ADT*.

3) Let *AN* be the <attribute name> of the <attribute definition>.

- 1 Rule deleted

4) If neither CONSTANT nor UPDATABLE is specified, then UPDATABLE is implicit.

- 1 Rule deleted

5) If <domain name> is specified, then let *D* be the domain identified by the <domain name>.

6) The data type of the attribute is

   Case:

   a) If <data type> is specified, then that data type.

   b) Otherwise, the data type of *D*.

7) If the data type of the attribute is character string, then the collation of the attribute is

   Case:

   a) If <collate clause> is specified, then the collation specified by that <collate clause>.

   b) If <domain name> is specified, then the collation of *D*, if any.

   c) Otherwise, the default collation of the character set of the attribute.

   Note: The character set of an attribute is determined by its data type.

8) If a <data type> is specified, then:

   a) Let *DT* be the <data type>.

   b) If DT is CHARACTER or CHARACTER VARYING and does not specify a <character set specification>, then the <character set specification> specified or implicit in the <schema character set specification> of the <schema definition> that created the schema identified by the <schema name> immediately contained in the <abstract data type name> of the containing <abstract data type definition> is implicit.

   c) If *DT* is a <character string type> that identifies a character set that specifies a <collate clause> and the <attribute definition> does not contain a <collate clause>, then the <collate clause> of the <character string type> is implicit in the <attribute definition>.

9) If <collate clause> is specified, then data type shall be a character string type.

10) If <domain name> is specified or implied, then <null clause> shall not be specified.

11) If <attribute definition> of an attribute is contained in an <abstract data type definition>, then:

   a) Let *AN* be the <attribute name> of the attribute.

   b) Let *DT* be the data type of the attribute.

   c) Let *ADTN* be the <abstract data type name> of the containing <abstract data type definition>.

   d) The containing <abstract data type definition> implicitly contains the <routine declaration>:

```
[ <encapsulation level> ]
    ACTOR FUNCTION AN ( ADT1 ADTN )
        RETURNS DT
```

   where the <encapsulation level> of the function *AN* is identical to the <encapsulation level> of attribute *AN*.

   This function is the observer function of attribute *AN*.

e) The containing <abstract data type definition> implicitly contains the <routine declaration>:

```
[ <encapsulation level> ]
    ACTOR FUNCTION AN ( ADT1 ADTN, ATTR DT )
        RETURNS ADTN
```

where the <encapsulation level> of the function *AN* is identical to the <encapsulation level> of attribute *AN*.

This function is the mutator function of attribute *AN*.

12) If the <attribute definition> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <domain name> shall include the descriptor of *D*. If the <attribute definition> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <domain name> shall include the descriptor of *D* or *S* shall include a <schema element> that creates the descriptor of *D*.

Access Rules

1) If <domain name> is specified, then the applicable privileges shall include USAGE on *D*.

2) If a <data type> is specified that is an abstract data type *A*, then the applicable privileges shall include USAGE on *A*.

3) If <null class name> is specified, then the applicable privileges shall include USAGE.

General Rules

1) An <attribute definition> defines an attribute of an abstract data type.

2) The <collate clause> specifies the default collating sequence for the attribute. If <collate clause> is not specified, then the default collating sequence is that used for comparisons of *Coercible* coercibility attribute, as defined in Subclause 8.2, "<comparison predicate>".

3) If the <attribute definition> specifies <data type>, then a data type descriptor is created that describes the data type of the attribute being defined.

4) The degree of the abstract data type *ADT* being defined in the containing <abstract data type definition> is increased by 1.

5) An attribute descriptor is created that describes the attribute being defined. The attribute descriptor includes:

   a) <attribute name>, the name of the attribute.

   b) Case:

      i) If the <attribute definition> specifies a <data type>, then the data type descriptor of the <data type> of the attribute.

      ii) Otherwise, the domain of the attribute.

   c) The ordinal position of the attribute, which is equal to the degree of *ADT*.

• 1 Rule deleted

d) The nullability characteristic of the attribute, determined according to the rules in Subclause 4.19, "Columns, fields, and attributes", and, if applicable, the null class of the attribute.

• 1 Rule deleted

e) The encapsulation level of the attribute.

f) Whether the attribute is READ, CONSTANT, or UPDATABLE.

g) If the <attribute definition> contains a <collate clause>, then the collation name.

6) Case:

a) If <domain name> is specified or implied, then the null class of this attribute is the null class of the domain specified by <domain name>.

b) If <null clause> is specified, then the null class of this attribute is the specified <null class name>.

c) Otherwise, the null class of this attribute is the general null class.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) An <attribute definition> shall not be specified.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.8 <field definition>

Function

Define a field of a row type.

Format

```
<field definition> ::=
       <field name> { <data type> | <domain name> }
       [ <collate clause> ]
       [ <null clause> ]
```

Syntax Rules

1) Let *RT* be the <row type> that simply contains a <field definition>.

2) The <field name> shall be different from the <field name> of any other <field definition> simply contained in *RT*.

3) If <domain name> is specified, then let *D* be the domain identified by the <domain name>.

4) The data type of the field is

   Case:

   a) If <data type> is specified, then that data type.

   b) If <domain name> is specified, then the data type of *D*.

5) If the data type of the field is character string, then the collation of the field is

   Case:

   a) If <collate clause> is specified, then the collation specified by that <collate clause>.

   b) If <domain name> is specified, then the collation of *D*, if any.

   c) Otherwise, the default collation of the character set of the field.

6) If a <data type> is specified, then:

   a) Let *DT* be the <data type>.

   b) If *DT* is CHARACTER or CHARACTER VARYING and does not specify a <character set specification>, then the <character set specification> specified or implicit in the <schema character set specification>.

   c) If *DT* is a <character string type> that identifies a character set that specifies a <collate clause> and the <field definition> does not contain a <collate clause>, then the <collate clause> of the <character string type> is implicit in the <field definition>.

   d) If <collate clause> is specified, then the data type shall be a character string type.

Access Rules

1) If a <data type> is specified that is an abstract data type *A*, then the applicable privileges shall include USAGE on *A*.

2) If <null class name> is specified, then the applicable privileges shall include USAGE.

General Rules

1) A <field definition> defines a field of a row type.

2) The <collate clause> specifies the default collating sequence for the field. If <collate clause> is not specified, then the default collating sequence is that used for comparisions of Coercible coercibility attribute, as defined in Subclause 8.2, "<comparison predicate>".

3) If the <field definition> specifies <data type>, then a data type descriptor is created that describes the data type of the field being defined.

4) The degree of the row type *RT* being defined in the simply containing <row type> is increased by 1.

5) A field descriptor is created that describes the field being defined. The field descriptor includes the following:

   a) The <field name>.

   b) If the <field definition> specifies <data type>, then the data type descriptor of the field and its null class.

   c) If the <field definition> contains a <collate clause>, then the <collation name> of the <collate clause>.

   d) The nullability characteristic of the field, as determined by the ruels of Subclause 4.19, "Columns, fields, and attributes".

   e) The field descriptor is included in the row type descriptor for *RT*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <field definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.9 <default clause>

Function

Specify the default for a column or domain.

Format

```
<default clause> ::=
        { DEFAULT | <assignment operator> } <default option>

<default option> ::=
        <literal>
      | <datetime value function>
      | USER
      | CURRENT_USER
      | SESSION_USER
      | SYSTEM_USER
      | CURRENT_PATH
      | NULL [ <left paren> <null state> <right paren> ]
```

- 1 alternative moved to Part 4

Syntax Rules

1) The subject data type of a <default clause> is the data type specified in the descriptor identified by the containing <column definition>, <domain definition>, <abstract data type definition>, <SQL parameter declaration>, <alter column definition>, or <alter domain statement>.

2) If USER is specified, then CURRENT_USER is implicit.

3) Case:

   a) If the subject data type of the <default clause> is an abstract data type, then <default option> shall specify NULL or <routine invocation>.

   b) If the subject data type of the <default clause> is a collection type, then <default option> shall specify NULL, <literal> or <routine invocation>.

4) Case:

   a) If a <literal> is specified, then:

      Case:

      i) If the subject data type is character string, then the <literal> shall be a <character string literal>. If the length of the subject data type is fixed, then the length in characters of the <character string literal> shall not be greater than the length of the subject data type. If the length of the subject data type is variable, then the length in characters of the <character string literal> shall not be greater than the maximum length of the subject data type. The <literal> shall have the same character repertoire as the subject data type.

ii) If the subject data type is bit string, then the <literal> shall be a <bit string literal> or a <hex string literal>. If the length of the subject data type is fixed, then the length in bits of the <bit string literal> or <hex string literal> shall not be greater than the length of the subject data type. If the length of the subject data type is variable, then the length in bits of the <bit string literal> or <hex string literal> shall not be greater than the maximum length of the subject data type.

iii) If the subject data type is binary string, then the <literal> shall be a <binary string literal> that has an even number of <hexit>s. The length in octets of the <binary string literal> shall not be greater than the maximum length of the subject data type.

iv) If the subject data type is exact numeric, then the <literal> shall be a <signed numeric literal> that simply contains an <exact numeric literal>. There shall be a representation of the value of the <literal> in the subject data type that does not lose any significant digits.

v) If the subject data type is approximate numeric, then the <literal> shall be a <signed numeric literal>.

vi) If the subject data type is an enumerated type, then the <literal> shall be an <enumeration literal> of that <enumerated type>.

vii) If the subject data type is datetime, then the <literal> shall be a <datetime literal> and shall contain the same <datetime field>s as the subject data type.

viii) If the subject data type is interval, then the <literal> shall be an <interval literal> and shall contain the same <interval qualifier> as the subject data type.

ix) If the subject data type is boolean, then the <literal> shall be a <boolean literal>.

x) If the subject data type is a collection type, then the data type of <literal> shall be that collection type.

****Editor's Note****

Paper X3H2-94-244/SOU-097 noted that the Subclause 5.3, "<literal>" appears not to premit literals of any collection type, so the preceding Rule is meaningless. See Possible Problem **411** in the Editor's Notes.

b) If CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH is specified, then the subject data type shall be character string with character set SQL_TEXT. If the length of the subject data type is fixed, then its length shall not be less than 128 characters. If the length of the subject data type is variable, then its maximum length shall not be less than 128 characters.

c) If <datetime value function> is specified, then the subject data type shall be datetime with the same datetime type as the datetime data type of the <datetime value function>.

d) A <null state>, if specified, shall be a valid null state of the null class of this column.

e) If <routine invocation> is specified, then the subject routine shall be a function and the subject data type shall be the same as the return data type of the function.

Access Rules

None.

General Rules

1) The default value inserted in the column descriptor, if the <default clause> is to apply to a column, or in the domain descriptor, if the <default clause> is to apply to a domain, or to the abstract data type descriptor, if the <default clause> is to apply to an abstract data type, is the <default option>.

2) If the subject data type is bit string with fixed length, the <default clause> specifies a <bit string literal>, and the length of the <bit string literal> is less than the fixed length of the column, then a completion condition is raised: *warning—implicit zero-bit padding*.

3) The default value of a column, an SQL variable, or an SQL parameter is

   Case:

   a) If the <column definition>, <SQL variable declaration>, or <parameter declaration> contains a <default clause>, then the value specified by that <default clause>.

   b) If the <column definition>, <SQL variable declaration>, or <parameter declaration> contains a <domain name> that identifies a domain descriptor that includes a <default clause>, then the value specified by that <default clause>.

   c) If the <column definition>, <SQL variable declaration>, or <parameter declaration> contains am <abstract data type name> that identifies an abstract data type descriptor that includes a <default clause>, then the value specified by that <default clause>.

   ANSI Only—caused by ISO changes not yet considered by ANSI

   d) If the default value is for a column *C* of a candidate row for insertion into or update of a derived table *DT* and *C* has a single counterpart column *CC* in a leaf generally underlying table of *DT*, then the default value of *CC* obtained by applying the General Rules of this Subclause.

   e) Otherwise, the general null value.

4) When a default value is required for a column, an SQL variable, or an SQL parameter, the default value for the column is derived from the <default option> as follows:

   a) If the column or domain descriptor contains NULL without a <null state>, then the general null value.

   b) If the <default option> specifies NULL with a <null state>, then the null value corresponding to that <null state>.

   c) If the column and domain descriptors both contain the null value, then the general null value.

   d) If the <default option> contains a <literal>, then

Case:

i) If the subject data type is numeric, then the numeric value of the <literal>.

ii) If the subject data type is character string with variable length, then the value of the <literal>.

iii) If the subject data type is character string with fixed length, then the value of the <literal>, extended as necessary on the right with <space>s to the length in characters of the subject data type.

iv) If the subject data type is bit string with variable length, then the value of the <literal>.

v) If the subject data type is bit string with fixed length, then the value of the <literal> extended as necessary on the right with 0-valued bits to the length of the subject data type.

vi) If the subject data type is binary string, then the value of the <literal>.

vii) If the subject data type is datetime or interval, then the value of the <literal>.

viii) If the subject data type is boolean, then the value of the <literal>.

ix) If the subject data type is an enumerated type, then the value of the <literal>.

x) If the subject data type is a collection type, then the value of the <literal>.

e) If the <default option> specifies CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH, then

Case:

i) If the subject data type is character string with variable length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH at the time that the default value is required.

ii) If the subject data type is character string with fixed length, then the value obtained by an evaluation of CURRENT_USER, SESSION_USER, CURRENT_PATH, or SYSTEM_USER at the time that the default value is required, extended as necessary on the right with <space>s to the length in characters of the subject data type.

f) If the <default option> contains a <datetime value function>, then the value of an evaluation of the <datetime value function> at the time that the default value is required.

g) If the <default option> contains a <routine invocation>, then the value of an invocation of that external function or SQL function at the time that the default value is required.

5) If the <default clause> is contained in an <SQL schema statement> and character representation of the <default option> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning—default value too long for information schema*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <default option> that specifies NULL shall not contain a <null state>.

   b) A <default option> shall not be a <routine invocation>.

   c) A <default option> shall not specify CURRENT_PATH.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <default option> shall not specify a <datetime value function>, SYSTEM_USER, SESSION_USER, or CURRENT_USER.

11.10 <table constraint definition>

Function

Specify an integrity constraint.

Format

```
<table constraint definition> ::=
       [ <constraint name definition> ]
       <table constraint> [ <constraint attributes> ]

<table constraint> ::=
         <unique constraint definition>
       | <referential constraint definition>
       | <check constraint definition>
```

Syntax Rules

1) If <constraint attributes> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

2) If <constraint name definition> is not specified, then a <constraint name definition> that contains an implementation-dependent <constraint name> is implicit. The assigned <constraint name> shall obey the Syntax Rules of an explicit <constraint name>.

Access Rules

None.

General Rules

1) A <table constraint definition> defines a table constraint.

2) A table constraint descriptor is created that describes the table constraint being defined. The table constraint descriptor includes the <constraint name> contained in the explicit or implicit <constraint name definition>.

   The table constraint descriptor includes an indication of whether the constraint is deferrable or not deferrable and whether the initial constraint mode of the constraint is *deferred* or *immediate*.

   Case:

   a) If <unique constraint definition> is specified, then the table constraint descriptor is a unique constraint descriptor that includes an indication of whether it was defined with PRIMARY KEY or UNIQUE, and the names of the unique columns specified in the <unique column list>.

   b) If <referential constraint definition> is specified, then the table constraint descriptor is a referential constraint descriptor that includes the names of the referencing columns specified in the <referencing columns> and the names of the referenced columns and referenced table specified in the <referenced table and columns>, the value of the <match type>, if specified, and the <referential triggered actions>, if specified.

c) If <check constraint definition> is specified, then the table constraint descriptor is a table check constraint descriptor that includes the <search condition>.

3) If the <table constraint> is a <check constraint definition>, then let *SC* be the <search condition> immediately contained in the <check constraint definition> and let *T* be the table name included in the corresponding table constraint descriptor; the table constraint is not satisfied if and only if

EXISTS (SELECT * FROM *T* WHERE NOT (*SC*))

is true.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall contain no <constraint name definition>.

11.11 <unique constraint definition>

Function

Specify a uniqueness constraint for a table.

Format

```
<unique constraint definition> ::=
        <unique specification> [ <left paren> <unique column list> <right paren> ]
      | UNIQUE ( VALUE )

<unique specification> ::=
        UNIQUE
      | PRIMARY KEY

<unique column list> ::= <column name list>
```

Syntax Rules

1) The data type of the column identified by any <column name> in the <unique column list> shall not be large object string.

2) Let *T* be the table identified by the containing <table definition> or <alter table statement>. Let *TN* be the <table name> of *T*.

3) If the <unique constraint definition> contains a non-empty <unique column list>, then let *UCL* be that <unique column list>.

   Case:

   a) If the <unique specification> specifies PRIMARY KEY, then let *SC* be the <search condition>:

      UNIQUE (SELECT *UCL* FROM *TN*)
      AND
      (*UCL*) IS NOT NULL

   b) Otherwise, let *SC* be the <search condition>:

      UNIQUE (SELECT *UCL* FROM *TN*)

4) If <unique specification> is specified but <unique column list> is not specified, then:

   a) The <unique specification> shall specify PRIMARY KEY.

   b) An empty <unique column list> is implicit.

   c) The following <check constraint definition> is implicit:

      CHECK ((SELECT COUNT(*) FROM *T*) <= 1)

5) If UNIQUE (VALUE) is specified, then let *SC* be the <search condition>

   UNIQUE (SELECT ROW * FROM *TN*)

   Note: A PRIMARY KEY without a <column name list> is commonly called a “nullary key”.

6) Each <column name> in the <unique column list> shall identify a stored column of *T*, and the same column shall not be identified more than once.

7) If any <column name> in the <unique column list> identifies a column whose data type is an abstract data type, then the <abstract data type definition> of that abstract data type shall contain an <equals clause> that does not specify NONE.

8) If the <unique specification> specifies PRIMARY KEY, then for each <column name> in the explicit or implicit <unique column list> for which NOT NULL is not specified, NOT NULL is implicit in the <column definition>.

9) A <table definition> shall specify at most one implicit or explicit <unique constraint definition> that specifies PRIMARY KEY.

10) If a <unique constraint definition> that specifies PRIMARY KEY is contained in an <add table constraint definition>, then the table identified by the <table name> immediately contained in the containing <alter table statement> shall not have a unique constraint that was defined by a <unique constraint definition> that specified PRIMARY KEY.

11) If UNIQUE (VALUE) is specified, then no other unique constraint descriptor shall be included in the base table descriptor of *T*.

12) The set of columns in the <unique column list> shall be distinct from the unique columns of any other unique constraint descriptor that is included in the base table descriptor of *T*.

13) If a <unique constraint definition> is contained in a <table definition> *TD* that specifies a <subtable clause>, then the <unique constraint definition> shall only specify UNIQUE and shall not specify PRIMARY KEY.

Access Rules

None.

General Rules

1) A <unique constraint definition> defines a unique constraint.

   Note: Subclause 10.7, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) Let *designated columns* be the columns identified by the <column name>s of the <unique column list>.

3) *T* is constrained to contain no rows that are duplicates with respect to the designated columns.

   Note: If the <unique column list> is empty, then all rows are duplicated, so *T* is constrained to hold not more than one row.

4) The unique constraint is not satisfied if and only if

   EXISTS (SELECT * FROM *TN* WHERE NOT (*SC*))

   is true.

****Editor's Note****

The preceding Rule was inserted here for alignment with SQL-92, but the Editor is uncertain whether it is appropriate, given the earlier General Rules in this Subclause. Guidance is requested.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <unique constraint definition> shall specify a <unique column list>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If PRIMARY KEY or UNIQUE is specified, then the <column definition> for each column whose <column name> is in the <unique column list> shall specify NOT NULL.

11.12 <referential constraint definition>

Function

Specify a referential constraint.

Format

```
<referential constraint definition> ::=
       FOREIGN KEY [ <left paren> <referencing columns> <right paren> ]
         <references specification>

<references specification> ::=
       REFERENCES [ PENDANT ] <referenced table and columns>
         [ MATCH <match type> ]
         [ <referential triggered action> ]

<match type> ::=
         FULL
       | PARTIAL

<referencing columns> ::=
       <reference column list>

<referenced table and columns> ::=
       <table name> [ <left paren> [ <reference column list> ] <right paren> ]

<reference column list> ::= <column name list>

<referential triggered action> ::=
         <update rule> [ <delete rule> ]
       | <delete rule> [ <update rule> ]

<update rule> ::= ON UPDATE <referential action>

<delete rule> ::= ON DELETE <referential action>

<referential action> ::=
         CASCADE
       | SET NULL [ <left paren> <null state> <right paren> ]
       | SET DEFAULT
       | RESTRICT
       | NO ACTION
```

Syntax Rules

1) The data type of the column identified by any <column name> in the <reference column list> shall not be large object string.

2) Let *referencing table* be the table identified by the containing <table definition> or <alter table statement>. Let *referenced table* be the table identified by the <table name> in the <referenced table and columns>. Let *referencing columns* be the column or columns identified by the <reference column list> in the <referencing columns> and let *referencing column* be one such column.

3) Case:

   a) If the <referenced table and columns> specifies a <reference column list>, then the set of column names of that <reference column list> shall be equal to the set of column names in the unique columns of a unique constraint of the referenced table. Let *referenced columns* be the column or columns identified by that <reference column list> and let *referenced column* be one such column. Each referenced column shall identify a column of the referenced table and the same column shall not be identified more than once.

   b) If the <referenced table and columns> does not specify a <reference column list>, then the table descriptor of the referenced table shall include a unique constraint that specifies PRIMARY KEY. Let *referenced columns* be the column or columns identified by the unique columns in that unique constraint and let *referenced column* be one such column. The <referenced table and columns> shall be considered to implicitly specify a <reference column list> that is identical to that <unique column list>.

4) The table constraint descriptor describing the <unique constraint definition> whose <unique column list> identifies the referenced columns shall indicate that the unique constraint is not deferrable.

5) The referenced table shall be a base table.

   Case:

   a) If the referencing table is a persistent base table, then the referenced table shall be a persistent base table.

   b) If the referencing table is a global temporary table, then the referenced table shall be a global temporary table.

   c) If the referencing table is a created local temporary table, then the referenced table shall be either a global temporary table or a created local temporary table.

   d) If the referencing table is a declared local temporary table, then the referenced table shall be either a global temporary table, a created local temporary table or a declared local temporary table.

6) If the referenced table is a temporary table with ON COMMIT DELETE ROWS specified, then the referencing table shall specify ON COMMIT DELETE ROWS.

7) Each referencing column shall identify a column of the referencing table, and the same column shall not be identified more than once.

8) The <referencing columns> shall contain the same number of <column name>s as the <referenced table and columns>. The *i*-th column identified in the <referencing columns> corresponds to the *i*-th column identified in the <referenced table and columns>. The data type of each referencing column shall be comparable to the data type of the corresponding referenced column.

9) If a <referential constraint definition> does not specify any <update rule>, then an <update rule> with a <referential action> of NO ACTION is implicit.

10) If a <referential constraint definition> does not specify any <delete rule>, then a <delete rule> with a <referential action> of NO ACTION is implicit.

11) If a <referential action> specifies <null state>, then each referencing column shall have the same null class and the null state identified by <null state> shall be in that null class.

12) If the <references specification> specifies PENDANT, then no <referential action> for that <referential constraint definition> shall specify SET DEFAULT.

13) If <references specification> specifies PENDANT, then <match type> shall not specify PARTIAL.

14) If <references specification> specifies PENDANT or <delete rule> specifies ON DELETE CASCADE, then the referencing table shall not be a table identified by the <table name> of a trigger descriptor whose <trigger event> specifies DELETE.

15) Let *T* be the referenced table. If the <referential constraint definition> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*. If the <referential constraint definition> is contained in a <schema definition> *S*, then the schema identified by the qualifier of the <table name> shall include the descriptor of *T* or *S* shall include a <schema element> that creates the descriptor of *T*.

Access Rules

1) The applicable privileges shall include REFERENCES for each referenced column. For a <references specification> specifying PENDANT, the applicable privileges for the <table name> shall also include DELETE.

General Rules

1) A <referential constraint definition> defines a referential constraint.

   Note: Subclause 10.7, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) Let R_f be the referencing columns and let R_t be the referenced columns in the referenced table *T*. The referencing table and the referenced table satisfy the referential constraint if and only if:

   Case:

   a) R_f and R_t are empty and

      0 < (SELECT COUNT (*) FROM *T*)

      is true.

   b) A <match type> is not specified and for each row of the referencing table, the <match predicate>

      R_f MATCH (SELECT R_t FROM *T*)

      is true.

   c) PARTIAL is specified and for each row of the referencing table, the <match predicate>

      R_f MATCH PARTIAL (SELECT R_t FROM *T*)

      is true.

   d) FULL is specified and for each row of the referencing table, the <match predicate>

      R_f MATCH FULL (SELECT R_t FROM *T*)

is true.

3) If PENDANT is specified, then:

   a) For a given row in the referencing table, let *pendant reference* designate an instance in which all R_f are non-null.

   b) Let *number of pendant paths* be the number of pendant references to the same referenced row in a referenced table from all referencing rows in all base tables.

   c) For every row in *T*, the number of pendant paths is equal to or greater than 1.

4) Case:

   a) If R_f and R_t are empty, then all rows in the referencing table are *matching rows* for the only row in *T*.

   b) If <match type> is not specified or if FULL is specified, then for a given row in the referenced table, let *matching rows* be all rows in the referencing table whose referencing column values equal the corresponding referenced column values for the referential constraint.

   c) If PARTIAL is specified, then:

      i) For a given row in the referenced table, let *matching rows* be all rows in the referencing table that have at least one non-null referencing column value and whose non-null referencing column values equal the corresponding referenced column values for the referential constraint.

      ii) For a given row in the referenced table, let *unique matching rows* be all matching rows for that given row that are matching rows only to the given row in the referenced table for the referential constraint. For a given row in the referenced table, let *non-unique matching rows* be all matching rows for that given row that are not unique matching rows for that given row for the referential constraint.

5) Case:

   a) If the cursor mode of the current SQL-transaction is *cascade on*, then for each row of the referenced table, its matching rows, unique matching rows, and non-unique matching rows are determined immediately prior to the execution of any <SQL procedure statement>. No new matching rows are added during the execution of that <SQL procedure statement> unless the association is a result of the direct update of a referencing row with pendant reference.

   b) If the cursor mode of the current SQL-transaction is *cascade off*, then matching rows, unique matching rows, and non-unique matching rows are evaluated immediately prior to the first execution of a <delete statement: positioned> or <update statement: positioned> in the current set-processing mode session of the current SQL-transaction. During execution of that <SQL procedure statement>, a referenced row may not enter into an association with a matching row that was not in effect immediately prior to the first execution of a <delete statement: positioned> or <update statement: positioned> in the current set-processing mode session of the current SQL-transaction unless the association is a result of the direct update of a referencing row with pendant reference.

The association between a referenced row and a non-unique matching row is dropped during the execution of that SQL-statement if the referenced row is either marked for deletion or updated to a distinct value on any referenced column that corresponds to a non-null referencing column. This occurs immediately after such a mark for deletion or update of the referenced row. Unique matching rows and non-unique matching rows for a referenced row are evaluated immediately after dropping the association between that referenced row and a non-unique matching row and immediately after creating or dropping any association between a referencing row with a pendant reference and that referenced row.

6) If a <delete rule> is specified and a row of the referenced table that has not previously been marked for deletion is marked for deletion, then

Case:

a) If <match type> is not specified or if FULL is specified, then

Case:

i) If the <delete rule> specifies CASCADE, then all matching rows are marked for deletion.

ii) If the <delete rule> specifies SET NULL, then in all matching rows each referencing column is set to the general null value.

iii) If the <delete rule> specifies SET DEFAULT, then in all matching rows each referencing column is set to the default value specified in the General Rules of Subclause 11.9, "<default clause>".

iv) If the <delete rule> specifies RESTRICT and there exists some matching row, then an exception condition is raised: *integrity constraint violation—restrict violation*.

b) If PARTIAL is specified, then

Case:

i) If the <delete rule> specifies CASCADE, then all unique matching rows are marked for deletion.

ii) If the <delete rule> specifies SET NULL, then in all unique matching rows each referencing column is set to the general null value.

iii) If the <delete rule> specifies SET DEFAULT, then in all unique matching rows each referencing column is set to the default value specified in the General Rules of Subclause 11.9, "<default clause>".

iv) If the <delete rule> specifies RESTRICT and there exists some unique matching row, then an exception condition is raised: *integrity constraint violation—restrict violation*.

Note: Otherwise, the <referential action> is not performed.

7) If an <update rule> is specified and a non-null value of a referenced column in the referenced table is updated to a value that is distinct from the current value of that column, then

Case:

a) If <match type> is not specified or if FULL is specified, then

Case:

i) If the <update rule> specifies CASCADE, then in all matching rows the referencing column that corresponds with the referenced column is updated to the new value of the referenced column.

ii) If the <update rule> specifies SET NULL, then

Case:

1) If <match type> is not specified, then in all matching rows the referencing column that corresponds with the referenced column is set to the general null value.

2) If <match type> specifies FULL, then in all matching rows each referencing column is set to the general null value.

iii) If the <update rule> specifies SET DEFAULT, then in all matching rows the referencing column that corresponds with the referenced column is set to the default value specified in the General Rules of Subclause 11.9, "<default clause>".

iv) If the <update rule> specifies RESTRICT and there exists some matching row, then an exception condition is raised: *integrity constraint violation—restrict violation*.

b) If PARTIAL is specified, then

Case:

i) If the <update rule> specifies CASCADE, then for each unique matching row that contains a non-null value in the referencing column *C1* that corresponds with the updated referenced column *C2*, *C1* is updated to the new value *V* of *C2*, provided that, in all updated rows in the referenced table that formerly had, in the same SQL-statement, that unique matching row as a matching row, the values in *C2* have all been updated to a value that is not distinct from *V*. Otherwise, an exception condition is raised: *triggered data change violation*.

Note: Because of the Rules of Subclause 8.2, "<comparison predicate>", on which the definition of "distinct" relies, the values in *C2* may have been updated to values that are not distinct, yet are not identical. Which of these non-distinct values is used for the cascade operation is implementation-dependent.

ii) If the <update rule> specifies SET NULL, then in all unique matching rows that contain a non-null value in the referencing column that corresponds with the updated column, that referencing column is set to the general null value.

iii) If the <update rule> specifies SET DEFAULT, then in all unique matching rows that contain a non-null value in the referencing column that corresponds with the updated column, that referencing column is set to the default value specified in the General Rules of Subclause 11.9, "<default clause>".

iv) If the <update rule> specifies RESTRICT and there exists some unique matching row, then an exception condition is raised: *integrity constraint violation—restrict violation*.

Note: Otherwise, the <referential action> is not performed.

8) If a referencing row with a pendant reference is inserted into a referencing table:

Case:

a) If no prior pendant reference to the referenced row exists, then the number of pendant paths for that referenced row is 1 and an association between the referencing row and that referenced row is created.

b) If the number of pendant paths for the referenced row is greater than or equal to 1, then the number of pendant paths for that referenced row is increased by 1 and an association between the referencing row and that referenced row is created.

9) If a referencing row with a pendant reference is marked for deletion from a referencing table, then the number of pendant paths for the referenced row is decreased by 1.

10) If a referencing column of a referencing table contained in a <references specification> specifying PENDANT is updated:

Case:

a) If the value of the referencing column is changed from a null value to a pendant reference, then cases under the General Rule beginning "If a referencing row with a pendant reference is inserted into a referencing table:" prevail.

b) If the value of the referencing column changes from a pendant reference to a null value, then the number of pendant paths for the referenced row is decreased by 1 and the association between the referencing row and that referenced row is dropped.

c) If the value of the referencing column changes from one pendant reference to another pendant reference, then the number of pendant paths for the referenced row of the former pendant reference is decreased by one, the association between the referencing row and that referenced row is dropped, and cases under the General Rule beginning "If a referencing row with a pendant reference is inserted into a referencing table:" prevail for the latter pendant reference.

11) Let *T* be the table identified by the <table name> of a <delete statement: searched> or an <update statement: searched>. If a mark for deletion of or update to *T* causes the number of pendant paths to some referenced row to decrease to 0, then a recursive search for each referenced row whose number of pendant paths is equal to 0 is effectively performed after all qualifying rows of *T* are marked for deletion or updated, and after all <referential action>s triggered by the marks for deletion or updates to *T* are performed. A recursive search comprises the following actions: If a referenced row with 0 pendant paths is found, it is marked for deletion; each subsequent search for another referenced row with 0 pendant paths is effectively performed after all <referential action>s triggered by the deletion of the previous referenced row are performed.

12) Case:

a) If the cursor mode of the current SQL-transaction is *cascade on*, then:

i) If any attempt is made within an SQL-statement to update some data item to a value that is distinct from the value to which that data item was previously updated within the same SQL-statement, then an exception condition is raised: *triggered data change violation*.

ii) If a data item in an object row is an <object column> of an <update statement: positioned> or <update statement: searched>, and there is any attempt within the same SQL-statement to delete the row containing that data item, then an exception condition is raised: *triggered data change violation*.

Note: Rows that are explicitly updated by an <update statement: positioned> or an <update statement: searched> are prohibited from later being (implicitly) deleted by a triggered PENDANT action.

b) If the cursor mode of the current SQL-transaction is *cascade off*, then:

i) If any attempt is made during the current set-processing mode session of the current SQL-transaction to update some data item to a value that is distinct from the value to which that data item was previously updated within the same set-processing mode session of the same SQL-transaction, then an exception condition is raised: *triggered data change violation*.

ii) If a data item in an object row is an <object column> of an <update statement: positioned> or <update statement: searched>, and there is any attempt within the same set-processing mode session of the same SQL-transaction to delete the row containing that data item, then an exception condition is raised: *triggered data change violation*.

Note: Rows that are explicitly updated by an <update statement: positioned> or an <update statement: searched> are prohibited from later being (implicitly) deleted by a triggered PENDANT action.

13) If an <update rule> attempts to update a row that has been deleted by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open or if a <delete rule> attempts to mark for deletion such a row, then a completion condition is raised: *warning—cursor operation conflict*.

14) Case:

a) If the cursor mode of the current SQL-transaction is *cascade on*, then all rows that are marked for deletion are effectively deleeted at the end of the SQL-statement, prior to the checking of any integrity constraints.

b) If the cursor mode of the current SQL-transaction is *cascade off*, then:

i) The deletion of all rows that are marked for deletion is effectively deferred until the execution of a <commit statement> or the execution of a <close statement>.

ii) All <referential action>s and pendant actions are effectively deferred until the execution of a <commit statement> or the execution of a <close statement>.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <references specification> shall not contain PENDANT.

b) If a <referenced table and columns> specifies a <left paren>, then it shall also specify a <reference column list>.

c) A <referential action> shall not be RESTRICT.

d) The data type of each referencing column shall be the same as the data type of the corresponding referenced column.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) A <references specification> shall not specify MATCH.

b) A <referential triggered action> shall not contain an <update rule>.

c) The order of the column names in a <reference column list> shall be the same as the order of column names of the corresponding unique constraint of the referenced table.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <referential constraint definition> shall not contain a <referential triggered action>.

11.13 <check constraint definition>

Function

Specify a condition for the SQL-data.

Format

```
<check constraint definition> ::=
       CHECK <left paren> <search condition> <right paren>
```

Syntax Rules

1) The <search condition> shall not contain a <target specification>.

2) The <search condition> shall not contain a <set function specification> that is not contained in a <subquery>.

3) If <check constraint definition> is contained in a <table definition> or <alter table statement>, then let *T* be the table identified by the containing <table definition> or <alter table statement>.

   Case:

   a) If *T* is a persistent base table, or if the <check constraint definition> is contained in a <domain definition> or <alter domain statement>, then no <table reference> generally contained in the <search condition> shall reference a temporary table.

   b) If *T* is a global temporary table, then no <table reference> generally contained in the <search condition> shall reference a table other than a global temporary table.

   c) If *T* is a created local temporary table, then no <table reference> generally contained in the <search condition> shall reference a table other than either a global temporary table or a created local temporary table.

   d) If *T* is a declared local temporary table, then no <table reference> generally contained in the <search condition> shall reference a persistent base table.

4) If the <check constraint definition> is contained in a <table definition> that defines a temporary table and specifies ON COMMIT PRESERVE ROWS or a <temporary table declaration> that specifies ON COMMIT PRESERVE ROWS, then no <subquery> in the <search condition> shall reference a temporary table defined by a <table definition> or a <temporary table declaration> that specifies ON COMMIT DELETE ROWS.

5) The <search condition> shall not generally contain a <datetime value function> or a <value specification> that is CURRENT_USER, SESSION_USER, SYSTEM_USER, or CURRENT_PATH.

6) The <search condition> shall not generally contain a <query specification> or a <query expression> that is possibly non-deterministic.

Access Rules

1) Let *TN* be any <table name> referenced in the <search condition>.

   Case:

   a) If a <column name> is contained in the <search condition>, then the applicable privileges shall include REFERENCES for each <column name> of the table identified by *TN* contained in the <search condition>.

   b) Otherwise, the applicable privileges shall include REFERENCES for at least one column of the table identified by *TN*.

General Rules

1) A <check constraint definition> defines a check constraint.

   Note: Subclause 10.7, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked. The General Rules that control the evaluation of a check constraint can be found in either Subclause 11.10, "<table constraint definition>", or Subclause 11.28, "<domain definition>", depending on whether it forms part of a table constraint or a domain constraint.

ANSI Only–SQL3

---

2) Case:

   a) If the <check constraint definition> is contained in a <table definition>, then it is applicable to that table.

   b) If the <check constraint definition> is contained in a <domain definition>, then it is applicable to every value required to be in that domain.

   c) If the <check constraint definition> is contained in an <abstract data type body>, then it is applicable to every instance of an abstract data type defined by or generated from that <abstract data type body>.

---

3) If the character representation of the <search condition> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning—search condition too long for information schema*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) The <search condition> contained in a <check constraint definition> shall not contain a <subquery>.

   b) The REFERENCES privilege is not required for <check constraint definition> access.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.14 <alter table statement>

Function

Change the definition of a table.

Format

```
<alter table statement> ::=
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
      ALTER <table type> TABLE <table name> <alter table action>
```

ISO Only--SQL3

```
      ALTER TABLE <table name> <alter table action>
```

```
<alter table action> ::=
        <add column definition>
      | <alter column definition>
      | <drop column definition>
      | <add supertable clause>
      | <drop supertable clause>
      | <add table constraint definition>
      | <drop table constraint definition>
```

ANSI Only—caused by ISO changes not yet considered by ANSI

Editor's Note
The Editor notes that the preceding ANSI syntax seems unnecessarily and confusingly different from the ANSI syntax in Subclause 11.5, "<table definition>", and should be updated similarly.

Syntax Rules

1) Let *T* be the table identified by the <table name>.

2) The schema identified by the explicit or implicit schema name of the <table name> shall include the descriptor of *T*.

ANSI Only—caused by ISO changes not yet considered by ANSI

3) If the specified <table type> is not TABLE, then it shall be the same as that in the descriptor for *T*.

---

4) The scope of the <table name> is the entire <alter table statement>.

5) *T* shall be a base table.

6) *T* shall not be a declared local temporary table.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by <table name>.

General Rules

1) The base table descriptor of *T* is modified as specified by <alter table action>.

2) If <add column definition> or <drop column definition> is specified, then the row type of *T* is the set of pairs (<field name>, <data type>) where <field column name> is the name of a column *C* of *T* and <data type> is the data type of *C*. This set of pairs contains one pair for each column of *T* in the order of their ordinal position in *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <alter table action> shall not be an <add supertable clause>.

   b) An <alter table action> shall not be a <drop supertable clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <alter table statement>.

11.15 <add column definition>

Function

Add a column to a table.

Format

```
<add column definition> ::=
      ADD [ COLUMN ] <column definition>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the table identified by the <table name> immediately contained in the containing <alter table statement>. The column defined by the <column definition> is added to *T*.

2) Let *C* be the column added to *T*. Every value in *C* is the default value for *C*.

   Note: The default value of a column is defined in Subclause 11.9, "<default clause>".

   Note: The addition of a column to a table has no effect on any existing <query expression> included in a view descriptor or <search condition> included in constraint descriptor because any implicit column references in these clauses are replaced by explicit column references when the clause is originally evaluated. See the Syntax Rules of Subclause 7.14, "<query expression>".

3) If *T* is the subject table in a trigger descriptor that contains an UPDATE trigger event with an implicit trigger column list, then *C* is included in the trigger column list of this trigger descriptor. The effect of adding *C* to the implicit trigger column list will be such that an update of *C* in *T* will cause the triggered actions to be executed.

4) For every table privilege descriptor that specifies *T* and a privilege of SELECT, UPDATE, INSERT or REFERENCES, a new column privilege descriptor is created that specifies *T*, the same action, grantor, and grantee, and the same grantability, and specifies the <column name> of the <column definition>.

5) In all other respects, the specification of a <column definition> in an <alter table statement> has the same effect as specification of the <column definition> in the <table definition> for *T* would have had. In particular, the degree of *T* is increased by 1 and the ordinal position of that column is equal to the new degree of *T* as specified in the General Rules of Subclause 11.6, "<column definition>".

6) If *T* is a subtable, then the descriptor of the column specified in the <column definition> defines an originally-defined column of *T*.

7) If *T* is a supertable, then an <add column definition>, without further Access Rule checking, is effectively performed for each of its subtables, thereby adding the column as an inherited column in these subtables.

****Editor's Note****
The preceding two rules have been identified as problematic. See Possible Problem **324** in the Editor's Notes.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <add column definition>.

11.16 <alter column definition>

Function

Change a column and its definition.

Format

```
<alter column definition> ::=
       ALTER [ COLUMN ] <column name> <alter column action>

<alter column action> ::=
         <set column default clause>
       | <drop column default clause>
       | <drop column domain clause>
```

Syntax Rules

1) Let *T* be the table identified in the containing <alter table statement>.

2) Let *C* be the column identified by the <column name>.

3) *C* shall be a column of *T*.

ISO Only—caused by ANSI changes not yet considered by ISO

4) *C* shall not have <column name> OID.

5) If <drop domain domain clause> is specified, then *C* shall be based on a domain.

Access Rules

None.

General Rules

1) The column descriptor of *C* is modified as specified by <alter column action>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <drop column domain clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <alter column definition>.

11.17 <set column default clause>

Function

Set the default clause for a column.

Format

```
<set column default clause> ::=
      SET <default clause>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *C* be the column identified by the <column name> in the containing <alter column definition>.

2) The default value specified by the <default clause> is placed in the column descriptor of *C*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <set column default clause>.

11.18 <drop column default clause>

Function

Drop the default clause from a column.

Format

```
<drop column default clause> ::=
       DROP DEFAULT
```

Syntax Rules

1) Let *C* be the column identified by the <column name> in the containing <alter column definition>.

2) The descriptor of *C* shall include a default value.

Access Rules

None.

General Rules

1) The default value is removed from the column descriptor of *C*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop column default clause>.

11.19 <drop column domain clause>

Function

Drop the domain from a column.

Format

```
<drop column domain clause> ::=
      DROP DOMAIN [ <constraint disposition> [ <constraint name list> ] ]
```

Syntax Rules

1) Let *D* be the domain of the column identified by the <column name> immdeiately contained in the containing <alter column definition> and let *DN* be the <domain name> of *D*.

2) If <constraint disposition> is not specified, then KEEP COLUMN CONSTRAINT is implicit.

3) If <constraint disposition> is specified and <constraint name list> is not specified, then ALL is implicit.

4) If a <constraint name list> is specified, then every <constraint name> in the <constraint name list> shall be the <constraint name> of a domain constraint descriptor included in the domain descriptor of *D*.

5) If ALL is specified or implicit, then it is equivalent to a <constraint name list> that contains each <constraint name> of every domain constraint descriptor included in the domain descriptor of *D*.

Access Rules

1) Let *UA* be the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by <table name>.

General Rules

1) Let *C* be the column descriptor of the column identified by the <column name> immediately contained in the containing <alter column definition>, let *T* be the table described by the table descriptor that includes *C*, and let *TN* be the table name of *T*. *C* is modified as follows:

   a) *DN* is removed from *C*. A copy of the data type descriptor of *D* is included in *C*.

   b) If *C* does not include a <default clause> and the domain descriptor of *D* includes a <default clause>, then a copy of the <default clause> of *D* is included in *C*.

   c) If DROP COLUMN CONSTRAINT is specified, then let the *excluded constraint list* be the <constraint name>s in the implicit or explicit <constraint name list>. If KEEP COLUMN CONSTRAINT is specified or implied, then let the *excluded constraint list* be the <constraint name> of each domain constraint descriptor included in the domain descriptor of *D* that does not occur in the implicit or explicit <constraint name list>.

For every domain constraint descriptor included in the domain descriptor of *D* whose <constraint name> is not contained in the excluded constraint list:

i) Let *TCD* be a <table constraint definition> consisting of a <constraint name definition> whose <constraint name> is implementation-dependent, whose <table constraint> is derived from the <check constraint definition> of the domain constraint descriptor by replacing every instance of VALUE by the <column name> of *C*, and whose <constraint attributes> are the <constraint attributes> of the domain constraint descriptor.

ii) If the applicable privileges of *UA* include all of the privileges necessary for *UA* to successfully execute the <add table constraint definition>

ALTER TABLE *TN* ADD *TCD*

then the following <table constraint definition> is effectively executed with a current <authorization identifier> of *UA*:

ALTER TABLE *TN* ADD *TCD*

d) If *C* does not include a collation and the <domain definition> of *D* includes a collation, then

i) Let *CCN* be the <collation name> of the collation.

ii) If the applicable privileges for *UA* contain USAGE on *CCN*, then *CCN* is added to *C* as the <collation name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall not contain any <drop column domain clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.20 <drop column definition>

Function

Destroy a column.

Format

```
<drop column definition> ::=
      DROP [ COLUMN ] <column name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement> and let *TN* be the name of *T*.

2) Let *C* be the column identified by the <column name> *CN*.

3) *C* shall be a column of *T* and *C* shall not be the only column of *T*.

4) If RESTRICT is specified, then *C* shall not be referenced in the <query expression> of any view descriptor or in the <search condition> of any constraint descriptor other than a table constraint descriptor that contains references to no other column and that is included in the table descriptor of *T*.

   Note: A <drop column definition> that does not specify CASCADE will fail if there are any references to that column resulting from the use of CORRESPONDING, NATURAL, SELECT * (except where contained in an exists predicate>), or REFERENCES without a <reference column list> in its <referenced table and columns>.

ISO Only—caused by ANSI changes not yet considered by ISO

---

5) *C* shall not have <column name> OID.

---

6) If RESTRICT is specified, then *CN* shall not be explicitly referenced in a <trigger definition>.

   Note: If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

7) If RESTRICT is specified, then *T* shall not be a supertable.

8) If *T* is a subtable, then the <column name> shall not specify a <column name> of an inherited column.

Access Rules

None.

General Rules

1) Let *TR* be the <trigger name> of any <trigger definition> that contains *CN*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

   DROP TRIGGER *TR*

2) If *T* is the subject table of a trigger descriptor *TD* that contains an UPDATE trigger event with an implicit trigger column list, then *C* is removed from the trigger column list of *TD*.

3) If *T* is a supertable, then let *ST* be the <table name> of any subtable of *T*. The following <alter table statement> is effectively executed without further Access Rule checking:

   ALTER TABLE *ST* DROP *CN* CASCADE

4) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE INSERT(*CN*), UPDATE(*CN*), SELECT(*CN*), REFERENCES(*CN*) ON TABLE *TN* FROM *A* CASCADE

5) If the column is not based on a domain, then its data type descriptor is destroyed.

6) The data associated with *C* is destroyed.

7) The descriptor of *C* is removed from the descriptor of *T*.

8) The identified column and its descriptor are destroyed.

9) The degree of *T* is reduced by 1. The ordinal position of all columns having an ordinal position greater than the ordinal position of *C* is reduced by 1.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop column definition>.

11.21 <add supertable clause>

Function

Make the table identified in the containing <alter table statement> a subtable of the table identified in the <add supertable clause>.

Format

```
<add supertable clause> ::=
      ADD <supertable clause>
```

Syntax Rules

1) Let *T* be the table identified in the containing <alter table statement>.

2) Let *TG* be the supertable identified by the <supertable name> in the <supertable clause>.

3) *T* shall not have *TG* as a subtable.

4) *TG* shall not have *T* as a direct subtable.

5) There shall be exactly one maximal supertable in the subtable family of *T* with the addition of *TG* to that family.

ANSI Only–SQL3

6) A row identifier is implicitly defined for *T*. For every table *ST* named in the <subtable clause>, a row identifier is implicitly defined for *ST*.

Access Rules

None.

General Rules

1) The name of *TG* is included as a direct supertable in the table descriptor of *T*.

2) The column descriptors of *TG* that are not already inherited by *T* are appended to the descriptor of *T* as inherited columns. For every <member renaming element> in the <supertable clause>, the <column name> of the inherited column identified by <supertable member name> is replaced by the <subtable member name>.

****Editor's Note****

Paper YOK-124/X3H2-93-214 noted that the preceding Rule is in need of rewriting because, among other things, the descriptor of *T* must contain the descriptor of *TG*, not merely the descriptors of the columns of *TG*. See Possible Problem **292** in the Editor's Notes.

3) For every newly inherited column in *T*, an <add column definition> is effectively performed for that column without further Access Rule checking.

ANSI Only–SQL3

4) If a row identifier is defined for a table *T*, then a new subtype *RDT* of the row identifier type is created. An implicit column of type *RDT* and name IDENTITY is created in *T*. *RDT* is a subtype of the row identifier defined for every supertable of *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <add supertable clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <add table constraint definition>.

11.22 <drop supertable clause>

Function

Amend the table identified in the containing <alter table statement> so that is not a subtable of the table identified in the <drop supertable clause>.

Format

```
<drop supertable clause> ::=
      DROP <supertable clause> <drop behavior>
```

Syntax Rules

1) Let *T* denote the table identified in the containing <alter table statement>.

2) Let *TG* denote the supertable identified by the <supertable name> in the <supertable clause>.

3) *T* shall be a direct subtable of *TG*.

Access Rules

None.

General Rules

1) For every column of *TG* that is not also inherited by *T* from some other supertable, a <drop column definition> is effectively performed for that column without further Access Rule checking. If <drop behavior> is specified, then the same <drop behavior>is implicitly specified in each of the <drop column definition>s.

****Editor's Note****

Paper YOK-124/X3H2-93-214 noted that the preceding Rule is in need of rewriting because, among other things, the descriptor of *T* must contain the descriptor of *TG*, not merely the descriptors of the columns of *TG*. See Possible Problem **292** in the Editor's Notes.

2) The name of the supertable is effectively dropped from the table descriptor of *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop supertable clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop table constraint definition>.

11.23 <add table constraint definition>

Function

Add a constraint to a table.

Format

```
<add table constraint definition> ::=
      ADD <table constraint definition>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement>.

2) The table constraint descriptor for the <table constraint definition> is included in the table descriptor for *T*.

3) Let *TC* be the table constraint added to *T*. If *TC* causes some column *CN* to be known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of the column descriptor of *CN* is changed to known not nullable.

   Note: The nullability characteristic of a column is defined in Subclause 4.19, "Columns, fields, and attributes".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain an <add table constraint definition>.

11.24 <drop table constraint definition>

Function

Destroy a constraint on a table.

Format

```
<drop table constraint definition> ::=
      DROP CONSTRAINT <constraint name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> in the containing <alter table statement>.

2) The <constraint name> shall identify a table constraint *TC* of *T*.

3) If *TC* is a unique constraint and there exists a referential constraint *RC* whose referenced table is *T* and whose referenced columns are the unique columns of *TC*, then *RC* is said to be *dependent on TC*.

ISO Only—caused by ANSI changes not yet considered by ISO

4) The constraint shall not be PRIMARY KEY on any column with <column name> OID.

5) If RESTRICT is specified, then no table constraint shall be dependent on *TC*.

   Note: If CASCADE is specified, then any such dependent object will be dropped by the effective execution of the <alter table statement> specified in the General Rules of this Subclause.

Access Rules

None.

General Rules

1) Let *TCN2* be the <constraint name> of any table constraint that is dependent on *TC* and let *T2* be the <table name> of the table descriptor that includes *TCN2*. The following <alter table statement> is effectively executed without further Access Rule checking:

   ALTER TABLE *T2* DROP CONSTRAINT *TCN2* CASCADE

2) The descriptor of *TC* is removed from the descriptor of *T*.

3) If *TC* causes some column *CN* to be known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of the column descriptor of *CN* is changed to possibly nullable.

   Note: The nullability characteristic of a column is defined in Subclause 4.19, "Columns, fields, and attributes".

4) The identified table constraint and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop table constraint definition>.

11.25 <drop table statement>

Function

Destroy a table.

Format

```
<drop table statement> ::=
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
DROP <table type> <table name> <drop behavior>
```

ISO Only--SQL3

```
DROP TABLE <table name> <drop behavior>
```

ANSI Only—caused by ISO changes not yet considered by ANSI

Editor's Note
The Editor notes that the preceding ANSI syntax seems unnecessarily and confusingly different from the ANSI syntax in Subclause 11.5, "<table definition>", and should be updated similarly.

Syntax Rules

1) Let *T* be the table identified by the <table name> and let *TN* be that <table name>.

2) The schema identified by the explicit or implicit schema name of the <table name> shall include the descriptor of *T*.

3) *T* shall be a base table.

ANSI Only—caused by ISO changes not yet considered by ANSI

4) If the specified <table type> is not TABLE, then it shall be the same as that in the descriptor for *T*.

5) *T* shall not be a declared local temporary table.

6) If RESTRICT is specified, then *T* shall not be referenced in the <query expression> of any view descriptor, the <search condition> of any table check constraint descriptor of any table other than *T*, or the table descriptor of the referenced table of any referential constraint descriptor of any table other than *T*.

   Note: If CASCADE is specified, then such referenced objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by *TN*.

General Rules

1) Let *ST* be the <table name> of any subtable of *T*. The following <drop table statement> is effectively executed without further Access Rule checking:

   DROP TABLE *ST* CASCADE

2) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE ALL PRIVILEGES ON *TN* FROM *A* CASCADE

3) The identified base table and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <drop table statement>.

11.26 <view definition>

Function

Define a viewed table.

Format

```
<view definition> ::=
      CREATE VIEW <table name> [ <left paren> <view column list> <right paren> ]
        AS <query expression>
        [ WITH [ <levels clause> ] CHECK OPTION ]

<levels clause> ::=
        CASCADED
      | LOCAL

<view column list> ::= <column name list>
```

Syntax Rules

1) The <query expression> shall not contain a <target specification>.

2) If a <view definition> is contained in a <schema definition> and the <table name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

3) The schema identified by the explicit or implicit schema name of the <table name> shall not include a table descriptor whose table name is <table name>.

4) No <table reference> generally contained in the <query expression> shall identify the viewed table defined by <view definition> nor any temporary view nor any declared local temporary table.

ISO Only–SQL3

5) If the <query expression> is inherently updatable, then the viewed table is an updatable table. Otherwise, it is not an inherently updatable table.

ANSI Only—caused by ISO changes not yet considered by ANSI

6) The viewed table is an inherently updatable table if and only if the <query expression> is inherently updatable.

7) If the <query expression> is a <query specification> that contains a <group by clause> or a <having clause> that is not contained in a <subquery>, then the viewed table defined by the <view definition> is a *grouped view*.

8) If any two columns in the table specified by the <query expression> have the same <column name>, or if any column of that table has an implementation-dependent name, then a <view column list> shall be specified.

9) The same <column name> shall not be specified more than once in the <view column list>.

10) The number of <column name>s in the <view column list> shall be the same as the degree of the table specified by the <query expression>.

11) No column in the table specified by <query expression> shall have a coercibility attribute of *No collating sequence*.

    Note: The coercibility attribute is described in Subclause 4.2.3, "Rules determining collating sequence usage".

    Note: The coercibility attribute for references to the column is defined in Subclause 6.3, "<item reference>".

12) If WITH CHECK OPTION is specified with no <levels clause>, then a <levels clause> of CASCADED is implicit.

13) If WITH LOCAL CHECK OPTION is specified, then the <query expression> shall not generally contain a <query expression> *QE* or a <query specification> *QS* that is *possibly non-deterministic* unless *QE* or *QS* is generally contained in a viewed table that is a root underlying table of the <query expression>.

    If WITH CASCADED CHECK OPTION is specified, then the <query expression> shall not generally contain a <query expression> or <query specification> that is *possibly non-deterministic*.

14) Let *V* be the view defined by the <view definition>. The underlying columns of every *i*-th column of *V* are the underlying columns of the *i*-th column of the <query expression> and the underlying columns of *V* are the underlying columns of the <query expression>.

Access Rules

1) If a <view definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <table name>.

General Rules

1) A view descriptor *VD* is created that describes *V*. The view descriptor includes the <table name>, the <query expression>, column descriptors taken from the table specified by the <query expression>, and an indication of whether WITH CHECK OPTION was specified. If a <view column list> is specified, then the <column name> of the *i*-th column of the view is the *i*-th <column name> in that <view column list>. Otherwise, the <column name>s of the view are the <column name>s of the table specified by the <query expression>.

ANSI Only–SQL3

The indication of whether the viewed table is a multiset table, a set table, or a list table are taken from the table specified by the <query expression>.

---

2) Let *VN* be the <table name>. Let *QE* be the <query expression>. If a <view column list> is specified, then let *VCL* be the <view column list> preceded by a <left paren> and followed by a <right paren>; otherwise, let *VCL* be the empty string.

   Case:

   a) When *VN* is immediately contained in some SQL-schema statement, it identifies the view descriptor *VD*.

   b) Otherwise, *VN* references the same table as the <table reference>:

      (*QE*) AS *VN VCL*

3) Let *A* be the <authorization identifier> that owns *V*.

4) For *i* ranging from 1 to the number of distinct leaf underlying tables of the <query expression> of *V*, let RT_i be the <table name>s of those tables. For every column *CV* of *V*:

   a) Let CRT_{ij}, for *j* ranging from 1 to the number of columns of RT_i that are underlying columns of *CV*, be the <column name>s of those columns.

   b) If *A* has REFERENCES(CRT_{ij}) for all *i* and for all *j*, and *A* has REFERENCES on some column of RT_i for all *i*, then a privilege descriptor is created that defines the privilege REFERENCES (*CV*) on *V* to *A*. That privilege is grantable if and only if the REFERENCES privileges on all of the columns CRT_{ij} are grantable. The grantor of that privilege descriptor is set to the special grantor value "_SYSTEM".

   c) If *A* has SELECT(CRT_{ij}) for all *i* and for all *j*, and *A* has SELECT on some column of RT_i for all *i*, then a privilege descriptor is created that defines the privilege SELECT (*CV*) on *V* to *A*. That privilege is grantable if and only if the SELECT privileges on all of the columns CRT_{ij} are grantable. The grantor of that privilege descriptor is set to the special grantor value "_SYSTEM".

5) Let *DT* be the derived table that is the result of the <query expression>. If *V* is inherently updatable, then for each leaf underlying table *LUT* of *DT*, if *DT* is fully inherently updatable with respect to *LUT*, then:

   a) A set of privilege descriptors is created that defines the privileges INSERT, UPDATE and DELETE on *V* that are applicable privileges on *LUT* to *A*.

   b) For each column *CV* of *V* that has a counterpart *CLUT* in *LUT*, a set of privilege descriptors is created that defines the privileges INSERT(*CV*) and UPDATE(*CV*) on *V*, where the privileges INSERT(*CLUT*) and UPDATE(*CLUT*) on *LUT* are applicable privileges to *A*, respectively.

   c) A privilege on *V* is grantable if and only if the corresponding privilege on *LUT* is grantable.

6) Let *V1* be a view. Let *V2* be the simply underlying table of the simply underlying table of the <query expression> of *V1*. If *V1* is inherently updatable, then let *S* be the set of *check conditions* of *V1* that is used for the validation of WITH CHECK OPTION. Each check condition of *S* is a *primary <search condition>* of a view. The primary <search condition> of a view is the <search condition> specified in the <where clause> of the <table expression> directly contained in the <query expression> of the view. If the <table expression> does not contain a <where clause>, then the view has an implicit primary search condition that is always true.

****Editor's Note****
V2 is not uniquely defined. See Possible Problem **413** in the Editor's Notes.

Case:

a) If *V2* is a base table and the descriptor of *V1* does not include WITH CHECK OPTION, then *S* is empty.

b) If *V2* is a base table and the descriptor of *V1* includes any form of WITH CHECK OPTION, then *S* consists of the primary <search condition> of *V1*.

c) If *V2* is a view and the descriptor of *V1* does not include WITH CHECK OPTION, then *S* is identical to the set of check conditions of *V2*.

d) If *V2* is a view and the descriptor of *V1* includes WITH CASCADED CHECK OPTION, then *S* consists of the primary <search condition> of *V1* and the primary <search condition> of each view that is a generally underlying table of *V1*.

e) If *V2* is a view and the descriptor of *V1* includes WITH LOCAL CHECK OPTION, then *S* consists of the primary <search condition> of *V1* and the set of check conditions of *V2*.

7) An *insert or update operation* is an <insert statement>, <update statement: positioned>, or <update statement: searched>. An insert or update operation on a view *V* is an insert or update operation whose immediately contained <table reference> identifies *V*.

8) During an insert or update operation on *V1*, every check condition in *S* is applied to every inserted or updated row. If any check condition is not satisfied, then an exception condition is raised: *with check option violation*.

9) Validation of a WITH CHECK OPTION constraint is effectively performed at the end of each operation.

10) If the character representation of the <query expression> cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning—query expression too long for information schema*.

11) Case:

a) If the cursor mode of the current SQL-transaction is *cascade off*, then the row validation for the WITH CHECK OPTION is effectively performed at the end of the <insert statement>, <update statement: positioned>, or <update statement: searched>.

b) If the cursor mode of the current SQL-transaction is *cascade on*, then the row validation for the WITH CHECK OPTION is effectively deferred until implicitly enacted by execution of a <commit statement> or implicitly enacted by execution of a <close statement>.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <levels clause>, but the effect shall be that defined for a <levels clause> of CASCADED.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) The <query expression> in a <view definition> shall be a <query specification>.

11.27 <drop view statement>

Function

Destroy a view.

Format

```
<drop view statement> ::=
      DROP VIEW <table name> <drop behavior>
```

Syntax Rules

1) Let *V* be the table identified by the <table name> and let *VN* be that <table name>. The schema identified by the explicit or implicit schema name of *VN* shall include the descriptor of *V*.

2) *V* shall be a viewed table.

3) If RESTRICT is specified, then *V* shall not be referenced in the <query expression> of any view descriptor, or the <search condition> of any assertion descriptor or constraint descriptor, or in any trigger descriptor.

   Note: If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the table identified by *VN*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE ALL PRIVILEGES ON *VN* FROM *A* CASCADE

2) The identified view and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop view statement>.

11.28 <domain definition>

Function

Define a domain.

Format

```
<domain definition> ::=
       CREATE DOMAIN <domain name> [ AS ] <data type>
         [ <default clause> ]
         [ <domain constraint>... ]
         [ <collate clause> ]
         [ <null clause> ]

<domain constraint> ::=
       [ <constraint name definition> ]
       <check constraint definition> [ <constraint attributes> ]
```

Syntax Rules

1) If a <domain definition> is contained in a <schema definition>, and if the <domain name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <domain name> shall not include a domain descriptor whose domain name is <domain name>, an abstract data type descriptor whose abstract data type name is <domain name>, nor a distinct type descriptor whose distinct type name is <domain name>.

3) If <data type> specifies a <character string type> and does not specify <character set specification>, then the character set name of the default character set of the schema identified by the implicit or explicit <schema name> of <domain name> is implicit.

4) If <data type> specifies a <character string type> that identifies a character set that has a default collation and the <domain definition> does not directly contain a <collate clause>, then the collation of the <character string type> is the implicit collation of the domain.

5) Let *D1* be some domain. *D1* is *in usage by* a domain constraint *DC* if and only if the <search condition> of *DC* generally contains the <domain name> either of *D1* or of some domain *D2* such that *D1* is in usage by some domain constraint of *D2*. No domain shall be in usage by any of its own constraints.

6) If <collate clause> is specified, then <data type> shall be a character string type.

7) For every <domain constraint> specified:

   a) If <constraint attributes> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

   b) If <constraint name definition> is not specified, then a <constraint name definition> that contains an implementation-dependent <constraint name> is implicit. The assigned <constraint name> shall obey the Syntax Rules of an explicit <constraint name>.

Access Rules

1) If a <domain definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <domain name>.

2) If <null class name> is specified, then the applicable privileges shall include USAGE.

General Rules

1) A <domain definition> defines a domain.

2) A data type descriptor is created that describes the data type of the domain being created.

3) A domain descriptor is created that describes the domain being created. The domain descriptor contains the name of the domain, the data type descriptor of the data type, the <collation name> of the <collate clause> if the <domain definition> contains a <collate clause>, the value of the <default clause> if the <domain definition> immediately contains <default clause>, and a domain constraint descriptor for every immediately contained <domain constraint>.

4) A privilege descriptor is created that defines the USAGE privilege on this domain to the <authorization identifier> of the schema or <module> in which the <domain definition> appears. This privilege is grantable if and only if the applicable privileges include a grantable REFERENCES privilege for each column reference included in the domain descriptor and a grantable USAGE privilege for each <domain name>, <collation name>, <character set name>, and <translation name> contained in the <search condition> of any domain constraint descriptor included in the domain descriptor. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".

5) Let *DSC* be the <search condition> included in some domain constraint descriptor *DCD*. Let *D* be the name of the domain whose descriptor includes *DCD*. Let *T* be the name of some table whose descriptor includes some column descriptor with column name *C* whose domain name is *D*. Let *CSC* be a copy of *DSC* in which every instance of the <general value specification> VALUE is replaced by *C*.

6) The domain constraint specified by *DCD* for *C* is not satisfied if and only if

   EXISTS (SELECT * FROM *T* WHERE NOT (*CSC*))

   is true.

   Note: Subclause 10.7, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

7) Case:

   i) If a <null class name> is specified, then the null class of this domain is <null class name>.

   ii) Otherwise, the null class of this domain is the general null class.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <domain definition> shall not contain a <null clause>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <domain definition>.

11.29 <alter domain statement>

Function

Change a domain and its definition.

Format

```
<alter domain statement> ::=
       ALTER DOMAIN <domain name> <alter domain action>

<alter domain action> ::=
         <set domain default clause>
       | <drop domain default clause>
       | <add domain constraint definition>
       | <drop domain constraint definition>
```

Syntax Rules

1) Let *D* be the domain identified by <domain name>. The schema identified by the explicit or implicit schema name of the <domain name> shall include the descriptor of *D*.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of <domain name>.

General Rules

1) The domain descriptor of *D* is modified as specified by <alter domain action>.

   Note: The changed domain descriptor of *D* is applicable to every column that is dependent on *D*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <alter domain statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.30 <set domain default clause>

Function

Set the default value in a domain.

Format

```
<set domain default clause> ::= SET <default clause>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *D* be the domain identified by the <domain name> in the containing <alter domain statement>.

2) The default value specified by the <default clause> is placed in the domain descriptor of *D*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <set domain default clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.31 <drop domain default clause>

Function

Remove the default clause of a domain.

Format

```
<drop domain default clause> ::= DROP DEFAULT
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> in the containing <alter domain statement>.

2) The descriptor of *D* shall contain a default value.

Access Rules

None.

General Rules

1) Let *C* be the set of columns whose column descriptors contain the domain descriptor of *D*.

2) For every column belonging to *C*, if the column descriptor does not already contain a default value, then the default value from the domain descriptor of *D* is placed in that column descriptor.

3) The default value is removed from the domain descriptor of *D*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <drop domain default clause>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.32 <add domain constraint definition>

Function

Add a constraint to a domain.

Format

```
<add domain constraint definition> ::=
      ADD <domain constraint>
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> in the <alter domain statement>.

2) Let *D1* be some domain. *D1* is *in usage by* a domain constraint *DC* if and only if the <search condition> of *DC* generally contains the <domain name> either of *D1* or of some domain *D2* such that *D1* is in usage by some domain constraint of *D2*. No domain shall be in usage by any of its own constraints.

Access Rules

None.

General Rules

1) The constraint descriptor of the <domain constraint> is added to the domain descriptor of *D*.

2) If *DC* causes some column *CN* to be known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of the column descriptor of *CN* is changed to known not nullable.

   Note: The nullability characteristic of a column is defined in Subclause 4.19, "Columns, fields, and attributes".

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <add domain constraint definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.33 <drop domain constraint definition>

Function

Destroy a constraint on a domain.

Format

```
<drop domain constraint definition> ::=
       DROP CONSTRAINT <constraint name>
         [ <constraint disposition> ]
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> *DN* in the containing <alter domain statement>.

2) Let *CD* be any column descriptor that includes *DN*, let *T* be the table described by the table desriptor that includes *CD*, and let *TN* be the <table name> of *T*.

3) If <constraint disposition> is not specified, then DROP COLUMN CONSTRAINT is implicit.

4) Let *DC* be the descriptor of the constraint identified by <constraint name>.

5) *DC* shall be included in the domain descriptor of *D*.

Access Rules

None.

General Rules

1) Let *UA* be the <authorization identifier> that owns the schema identified by the <schema name> of a table identified by *TN*.

2) If KEEP COLUMN CONSTRAINT is specified, then for every column *C* that is based on domain *D*:

   a) Let *TCD* be a <table constraint definition> comprising a <constraint name definition> whose <constraint name> is implementation-dependent, whose <table constraint> is derived from the <check constraint definition> of *DC* by replacing every instance of VALUE by the <column name> of *C*, and whose <constraint attributes> are the <constraint attributes> of *DC*.

   b) If the applicable privileges of *UA* include all of the privileges necessary for *UA* to successfully execute the <add table constraint definition>

      ALTER TABLE *TN* ADD *TCD*

      then the following <table constraint definition> is effectively executed with a current <authorization identifier> of *UA*:

      ALTER TABLE *TN* ADD *TCD*

3) The constraint descriptor *DC* is removed from the domain descriptor of *D*.

4) If *DC* causes some column *CN* to be known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of the column descriptor of *CN* is changed to possibly nullable.

   Note: The nullability characteristic of a column is defined in Subclause 4.19, "Columns, fields, and attributes".

5) The constraint *DC* and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <constraint disposition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <drop domain constraint definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.34 <drop domain statement>

Function

Destroy a domain.

Format

```
<drop domain statement> ::=
       DROP DOMAIN <domain name> <drop behavior>
         [ <constraint disposition> [ <constraint name list> ] ]

<constraint disposition> ::=
         KEEP COLUMN CONSTRAINT
       | DROP COLUMN CONSTRAINT
```

Syntax Rules

1) Let *D* be the domain identified by <domain name> and let *DN* be that <domain name>. The schema identified by the explicit or implicit schema name of *DN* shall include the descriptor of *D*.

2) If RESTRICT is specified, then *D* shall not be referenced by any column descriptor, in the <query expression> of any view descriptor, or in the <search condition> of any constraint descriptor.

3) If RESTRICT is specified, then <constraint disposition> shall not be specified.

4) If CASCADE is specified and <constraint disposition> is not specified, then KEEP COLUMN CONSTRAINT is implicit.

5) If <constraint disposition> is specified and <constraint name list> is not specified, then ALL is implicit.

6) If a <constraint name list> is specified, then every <constraint name> in the <constraint name list> shall be the <constraint name> of a domain constraint descriptor included in the domain descriptor of *D*.

7) If ALL is specified or implicit, then it is equivalent to a <constraint name list> that contains the <constraint name> of every domain constraint descriptor included in the domain descriptor of *D*.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the domain identified by *DN*. Let *UA* be the <authorization identifier> of the current SQL-session.

General Rules

1) Let *C* be any column descriptor that includes *DN*, let *T* be the table described by the table descriptor that includes *C*, and let *TN* be the table name of *T*. *C* is modified as follows:

   a) *DN* is removed from *C*. A copy of the data type descriptor of *D* is included in *C*.

   b) If *C* does not include a <default clause> and the domain descriptor of *D* includes a <default clause>, then a copy of the <default clause> of *D* is included in *C*.

   c) If DROP COLUMN CONSTRAINT is specified, then let the *excluded constraint list* be the <constraint name>s in the implicit or explicit <constraint name list>. If KEEP COLUMN CONSTRAINT is specified or implied, then let the *excluded constraint list* be the <constraint name> of each domain constraint descriptor included in the domain descriptor of *D* that does not occur in the implicit or explicit <constraint name list>.

      For every domain constraint descriptor included in the domain descriptor of *D* whose <constraint name> is not contained in the excluded constraint list:

      i) Let *TCD* be a <table constraint definition> consisting of a <constraint name definition> whose <constraint name> is implementation-dependent, whose <table constraint> is derived from the <check constraint definition> of the domain constraint descriptor by replacing every instance of VALUE by the <column name> of *C*, and whose <constraint attributes> are the <constraint attributes> of the domain constraint descriptor.

      ii) If the applicable privileges of *UA* include all of the privileges necessary for *UA* to successfully execute the <add table constraint definition>

         ALTER TABLE *TN* ADD *TCD*

         then the following <table constraint definition> is effectively executed with a current <authorization identifier> of *UA*:

         ALTER TABLE *TN* ADD *TCD*

   d) If *C* does not include a collation and the <domain definition> of *D* includes a collation, then

      i) Let *CCN* be the <collation name> of the collation.

      ii) If the applicable privileges for *UA* contain USAGE on *CCN*, then *CCN* is added to *C* as the <collation name>.

2) If *T* is a supertable, then all subtables of *T* are destroyed.

****Editor's Note****

The preceding Rule causes all subtables of a supertable to be destroyed if a domain used by any column in the supertable is dropped. That seems an unusually harsh penalty when the data type and other attributes of the domain are "adopted" by the column in the supertable—couldn't the same "adoption" be made by the corresponding columns in the subtables? Guidance is requested.

3) The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE USAGE ON DOMAIN *DN* FROM *UA* CASCADE

****Editor's Note****

The preceding Rule and the following one seem to differ in one important respect. The following Rule (currently ANSI-only) revokes every privilege on the object granted by "_SYSTEM" to the owner/creator of the object, whereas the preceding Rule (currently ISO-only) revokes every privilege on the object granted by "_SYSTEM" to the destroyer of the object. In SQL2, the destroyer of an object must, of necessity, be the creator of the object (that is because only the owner of a schema can create and destroy objects in that schema). However, a possible SQL3 enhancement might be the definition of "schema privileges" that permit actors other than the owner of a schema to create and destroy objects in that schema—even to the point that an actor other than the creator of an object might be permitted to destroy the object.

The author of LON-070, point H, correctly notes that Subclause 11.40, "<drop collation statement>", already has a Rule of this (preceding) type. Putting such Rules in place everywhere proposed by LON-070, point H, is fine, but we must remember to revise the Rules if any SQL3 "schema privilegs" proposal is ever accepted.

This comment applies to this Subclause as well as to Subclause 11.25, "<drop table statement>", Subclause 11.27, "<drop view statement>", Subclause 11.20, "<drop column definition>", Subclause 11.40, "<drop collation statement>", Subclause 11.38, "<drop character set statement>", and Subclause 11.42, "<drop translation statement>".

The Rule (which ANSI deleted in favor of the ISO rule above):

For every privilege descriptor with a grantor of "_SYSTEM" that specifies USAGE on *D*, the following <revoke statement> is effectively executed as if the <authorization identifier> of the current SQL-session were "_SYSTEM" and without Access Rule checking:

REVOKE USAGE ON DOMAIN *DN* FROM *GRANTEE* CASCADE

where *GRANTEE* is the grantee in the privilege descriptor.

4) The identified domain is destroyed by destroying its descriptor and its data type descriptor.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <constraint disposition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <drop domain statement>.

11.35 <null class definition>

Function

Define a null class.

Format

```
<null class definition> ::=
      CREATE NULL CLASS <null class name> [ AS ]
          ( <null state list> )

<null state list> ::=
      <null state> [ { <comma> <null state> }... ]

<null state> ::= <identifier>
```

Syntax Rules

1) Case:

   a) If a <null class definition> is contained in a <schema definition>, and if the <null class name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If a <null class definition> is contained in a <module>, and if the <null class name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <module>.

2) The <null class name> shall be different from the <null class name> of any other null class in the containing schema.

3) In a <null state list>, each <null state> shall be different from any other <null state> of the <null state list>.

4) A <null state list> defines an ordering of null states. Each <null state> yields a different null state value. The predefined order relation between null state values follows the order of corresponding position numbers. The position number of the value of the first listed <null state> is 2. The position number of each other null state value is one more than the position number of its predecessor in the list.

Access Rules

1) If a <null class definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <null class name>.

General Rules

1) A privilege descriptor is created that defines the USAGE privilege on this null class to the <authorization identifier> of the schema or <module> in which the <null class definition> appears. This privilege is grantable. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".

2) The possible null states of the null class are the general null value (effectively having position number one) together with null states identified in the <null state list>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <null class definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.36 <drop null class statement>

Function

Destroy a null class.

Format

```
<drop null class statement> ::=
      DROP NULL CLASS <null class name>
```

Syntax Rules

1) Let *N* denote the null class identified by <null class name> and let *NN* be the <null class name>.

2) *N* shall not be the null class of any domain or column.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the null class identified by *NN*.

General Rules

1) For every privilege descriptor with a grantor of "_SYSTEM" that specifies USAGE on *N*, the following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without Access Rule checking:

   REVOKE USAGE ON *NN* FROM *GRANTEE* CASCADE

   where *GRANTEE* is the grantee in the privilege descriptor.

2) The descriptor of *N* is destroyed and *N* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop null class statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.37 <character set definition>

Function

Define a character set.

Format

```
<character set definition> ::=
       CREATE CHARACTER SET <character set name> [ AS ]
         <character set source>
         <form-of-use specification>
         [ <collate clause> | <limited collation definition> ]

<character set source> ::=
         GET <existing character set name>
             [ <plus sign> <character set source> ]
       | <left paren> <character list> <right paren>

<existing character set name> ::=
         <standard character repertoire name>
       | <implementation-defined character repertoire name>
       | <schema character set name>

<form-of-use specification> ::= <identifier>

<schema character set name> ::= <character set name>

<limited collation definition> ::=
       COLLATION FROM <collation source>

<character list> ::=
       <character specification> [ { <comma> <character specification> }... ]

<character specification> ::=
         <character string literal>
       | <ISO 10646 position>
       | <ISO 10646 character name>

<ISO 10646 position> ::=
       <val> [ <sep> <val> [ <sep> <val> [ <sep> <val> ] ] ]

<val> ::= <unsigned numeric literal>

<sep> ::= <ampersand>

<ISO 10646 character name> ::= !! See the Syntax Rules
```

Syntax Rules

1) If a <character set definition> is contained in a <schema definition> and if the <character set name> immediately contained in the <character set definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <character set name> shall not include a character set descriptor whose character set name is <character set name>.

3) The <standard character repertoire name>s and <implementation-defined character repertoire name>s that are supported are implementation-defined. Each character set identified by a <standard character repertoire name> or by an <implementation-defined character repertoire name> shall have associated with it a privilege descriptor that was effectively defined by the <grant statement>

   GRANT USAGE ON CHARACTER SET *CS* TO PUBLIC

   where *CS* is the <standard character repertoire name> or <implementation-defined character repertoire name>.

4) A <character string literal> directly contained in a <character specification> shall have length 1.

5) A <schema character set name> shall identify some character set.

   Note: If the <character set definition> is not specified in a <schema definition>, then <collate clause> cannot be specified, because this "circular reference" is permitted only when the <character set definition> and the <collation definition> are defined "simultaneously" in a single <schema definition>.

6) An <ISO 10646 position> shall be a valid code that identifies a character position in ISO/IEC DIS 10646.

7) An <ISO 10646 character name> shall be the name of a character as given in ISO/IEC DIS 10646 except that <space> characters are replaced by <underscore>s in the name.

8) If the <character set definition> is not specified in a <schema definition>, then <collate clause> shall not be specified.

9) If neither <collate clause> nor <limited collation definition> is specified, then the following <limited collation definition> is implicit:

   COLLATION FROM DEFAULT

10) If <limited collation definition> is specified, then <collation source> shall not be an <internal collation source>.

Access Rules

1) If a <character set definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <character set name>.

2) The applicable privileges for the <existing character set name> shall include USAGE.

General Rules

1) A <character set definition> defines a character set.

2) A character set descriptor is created for the defined character set.

3) The character set that is created contains every character in each of the character sets identified by <existing character set name>s, if specified, and in the <character set list>, if specified. Any redundant duplicate characters are deleted from the created character set.

4) If ONE_OCTET is specified, then the resulting character set is represented with 1 octet per character. If TWO_OCTET is specified, then the resulting character set is represented with 2 octets per character. If FOUR_OCTET is specified, then the resulting character set is represented with 4 octets per character.

5) A privilege descriptor is created that defines the USAGE privilege on this character set to the <authorization identifier> of the schema or <module> in which the <character set definition> appears. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM". This privilege is grantable.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <character set source> shall not contain a <plus sign>.

   b) A <character set source> shall not contain a <character list>.

   c) A <character set definition> shall not contain a <form-of-use specification>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) In conforming Intermediate SQL language, <collation source> shall specify DEFAULT.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not specify any <character set definition>.

11.38 <drop character set statement>

Function

Destroy a character set.

Format

```
<drop character set statement> ::=
      DROP CHARACTER SET <character set name>
```

Syntax Rules

1) Let *C* be the character set identified by the <character set name> and let *CN* be the name of *C*.

2) The schema identified by the explicit or implicit schema name of *CN* shall include the descriptor of *C*.

3) *C* shall not be referenced in the <query expression> of any view descriptor or in the <search condition> of any constraint descriptor, nor be included in any collation descriptor or translation descriptor or schema descriptor, nor be referenced in the <module character set specification> of any module, nor be included in any column's or any domain's data type descriptor.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the character set identified by *C*.

General Rules

1) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE USAGE ON CHARACTER SET *CN* FROM *A* CASCADE

2) The descriptor of *C* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall contain no <drop character set statement>.

11.39 <collation definition>

Function

Define a collating sequence.

Format

```
<collation definition> ::=
       CREATE COLLATION <collation name> FOR <character set specification>
         FROM <collation source>
           [ <pad attribute> ]

<pad attribute> ::=
         NO PAD
       | PAD SPACE

<collation source> ::=
         <collating sequence definition>
       | <translation collation>
       | <collation dictionary specification>
       | <collation routine specification>

<collation routine specification> ::=
         ROUTINE <left paren> <implementation-defined routine name>
             <left paren> <params> <right paren> <right paren>

<params> ::= !! Not yet defined

<implementation-defined routine name> ::= !! See the Syntax Rules



<collating sequence definition> ::=
         <external collation>
       | <schema collation name>
       | <internal collation source>
       | DESC <left paren> <collation name> <right paren>
       | DEFAULT

<translation collation> ::=
       TRANSLATION <translation name>
           [ THEN COLLATION <collation name> ]

<collation dictionary specification> ::=
       DICTIONARY <dictionary name> [ <plus sign> <dictionary name> ]

<dictionary name> ::=
       <quote> <implementation-defined dictionary name> <quote>

<implementation-defined dictionary name> ::= !! See the Syntax Rules

<external collation> ::=
       EXTERNAL <left paren> <quote> <external collation name> <quote> <right paren>

<schema collation name> ::= <collation name>

<external collation name> ::=
         <standard collation name>
```

```
       | <implementation-defined collation name>

<standard collation name> ::= <collation name>

<implementation-defined collation name> ::= <collation name>

<internal collation source> ::= <left paren> <collation options> <right paren>

<collation options> ::=
       <collation option> [ { <comma> <collation option> }... ]

<collation option> ::=
         USING <left paren> <collating basis> <right paren>
       | SEQUENCE <left paren> <enumerated collating sequence> <right paren>
       | MODIFY <left paren> <collating modifiers> <right paren>
       | WHEN NOT FOUND
           { IGNORE | MAX | MIN }

<enumerated collating sequence> ::=
       <collating chars> [ { <comma> <collating chars> }... ]

<collating chars> ::=
         <character specification>
       | <character range>

<character range> ::=
       <character specification> <minus sign> <character specification>

<collating modifiers> ::=
         <collating modifier> [ { <comma> <collating modifier> }... ]

<collating modifier> ::=
       <collating chars>
         { <less than operator> | <greater than operator> | <equals operator> }
       <collating chars>

<collating basis> ::=
       <collating foundation> [ { <plus sign> <collating foundation> }... ]

<collating foundation> ::=
       <collating sequence definition>
```

Note that the BNF nonterminal "<params>" is not defined.

Syntax Rules

1) If a <collation definition> is contained in a <schema definition> and if the <collation name> immediately contained in the <collation definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <collation name> shall not include a collation descriptor whose collation name is <collation name>.

3) A <standard collation name> shall be the name of a collation defined by a national or international standard. An <implementation-defined collation name> shall be the name of a collation that is implementation-defined.

4) The <standard collation name>s and <implementation-defined collation name>s that are supported are implementation-defined. Each collation identified by a <standard collation name> or by an <implementation-defined collation name> shall have associated with it a privilege descriptor that was effectively defined by the <grant statement>

   GRANT USAGE ON COLLATION *COLL* TO PUBLIC

   where *COLL* is the <standard collation name> or <implementation-defined collation name>.

5) A collating sequence specified by <external collation name> or <schema collation name> shall be a collating sequence that is defined for the character repertoire of the character set with which the <collation source> is associated.

6) A <schema collation name> shall be the name of a collating sequence that is defined in the schema identified by the explicit or implicit <schema name>.

7) An <implementation-defined routine name> shall be the name of an implementation-defined collation routine.

8) An <implementation-defined dictionary name> shall be the name of an implementation-defined dictionary of collation information. The structure of the dictionary is implementation-defined.

9) Each <character representation> that is contained in <collating chars> shall be a character in the character repertoire identified by <character set specification>.

10) Each collating sequence specified by <external collation> or <schema collation name> shall be a collating sequence that is defined for the character repertoire identified by <character set specification>.

11) No <collation options> shall directly contain USING, SEQUENCE, MODIFY, or WHEN NOT FOUND more than once.

12) In <collation options>, MODIFY shall not be specified first.

13) In <collation options>, if USING is specified, then SEQUENCE shall not be specified.

14) A <character range> *i*–*j* is equivalent to the <collating chars> *i*,...,*j*, where every character in the repertoire that appears between *i* and *j* is included in the ellipsis, in the order that the characters appear in the character repertoire.

15) If a <collation definition> does not specify <pad attribute>, then

    Case:

    a) If a <collating sequence definition> is specified that contains a <collation name> that identifies a collation for which the <collation definition> specifies NO PAD, then NO PAD is implicit.

    b) Otherwise, PAD SPACE is implicit.

16) If NO PAD is specified, then the collation is said to have the NO PAD attribute. If PAD SPACE is specified, then the collation is said to have the PAD SPACE attribute.

17) If <translation collation> is specified, then let *T* be the translation named by <translation name>. Let *C1* be the collation being defined by the <collation definition>. The source character set of *T* shall be the same as the character set of *C1*.

18) If THEN COLLATION <collation name> is specified, then let *C2* be the collation named by <collation name> in THEN COLLATION <collation name>. The target character set of *T* shall be identical to the character set of *C2*.

19) If <translation name> is specified and the <collation definition> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <translation name> shall include the descriptor of *T*. If the <collation definition> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <translation name> shall include the descriptor of *T* or *S* shall include a <schema element> that creates the descriptor of *T*.

20) Let *C* be a collation identified by any <collation name> contained in <collation source>. If the <collation definition> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <collation name> shall include the descriptor of *C*. If the <collation definition> is contained in a <schema definition> *S*, then the explicit or implicit qualifier of the <collation name> shall include the descriptor of *C* or *S* shall include a <schema element> that creates the descriptor of *C*.

Access Rules

1) If a <collation definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <collation name>.

2) The applicable privileges shall include USAGE on *C*.

3) If <translation name> is specified, then the applicable privileges shall include USAGE.

General Rules

1) A <collation definition> defines a collating sequence.

2) DEFAULT specifies that the collation is to be performed using the order of characters as they appear in the character repertoire.

3) If DESC is specified, then the collation is the reverse of that specified by <collation name>.

4) If a <collation dictionary specification> specifies more than one dictionary, then the dictionaries are effectively concatenated.

5) In a <collating modifier>, <less than operator> means that the <collating chars> on the left of the <less than operator> collate immediately preceding the <collating chars> on the right of the <less than operator>.

6) In a <collating modifier>, <greater than operator> means that the <collating chars> on the left of the <greater than operator> collate immediately following the <collating chars> on the right of the <greater than operator>.

7) In a <collating modifier>, <equals operator> means that the <collating chars> on the left of the <equals operator> collate the same as the <collating chars> on the right of the <equals operator>.

8) If <internal collation source> is specified, then:

   a) If USING is specified, then the collation is the <collating basis>.

b) If SEQUENCE is specified, then the collation is the <enumerated collating sequence>.

c) If MODIFY is specified, then the collation is modified by applying <collating modifiers> to the collation.

d) If WHEN NOT FOUND is specified, then

Case:

i) If IGNORE is specified, then characters for which a collation is not otherwise given are ignored when using this collation.

ii) If MAX is specified, then characters for which a collation is not otherwise given are collated greater than any character for which a collation is given, but their collation is otherwise implementation-dependent.

iii) If MIN is specified, then characters for which a collation is not otherwise given are collated less than any character for which a collation is given, but their collation is otherwise implementation-dependent.

e) In a <collating basis>, multiple <collating foundation>s identify collations that are to be applied sequentially in the order that they are specified.

9) A privilege descriptor is created that defines the USAGE privilege on this collation to the current <authorization identifier>. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".

10) This privilege descriptor is grantable if and only if the USAGE privilege for the current <authorization identifier> on the <character set name> contained in the <collation definition> is also grantable and if the USAGE privilege for the current <authorization identifier> on the <translation name> contained in the <translation collation>, if present, is also grantable.

11) If <translation collation> is specified, then

Case:

a) If THEN COLLATION <collation name> is specified, then let *C2* be the collating sequence named by the <collation name> in THEN COLLATION <collation name>. The collating sequence defined is obtained by effectively translating a character string using *T*, then applying the collating sequence of *C2* to the result.

b) Otherwise, the collating sequence defined is obtained by effectively translating a character string using *T*, then applying the default collating sequence for the target character set of *T*.

12) If <external collation> is specified, then the collating sequence defined is that given by:

a) If <standard collation name> is specified, then the national or international standard collation.

b) Otherwise, the implementation-defined collation.

13) A collation descriptor is created for the defined collation.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <collation source> shall not be a <collation dictionary specification>.

   b) A <collation source> shall not be a <collation routine specification>.

   c) A <collation sequence definition> shall not be an <internal collation source>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <collation definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.40 <drop collation statement>

Function

Destroy a collating sequence.

Format

```
<drop collation statement> ::=
      DROP COLLATION <collation name>
        <drop behavior>
```

****Editor's Note****

The CASCADE capability for <drop collation statement> has been deferred 'til SQL3 because of certain implications: dropping a collation CASCADE will cause columns using that collation to be dropped, including the data in the column. This problem requires further thought.

Syntax Rules

1) Let *C* be the collating sequence identified by the <collation name> and let *CN* be the name of *C*.

2) The schema identified by the explicit or implicit schema name of *CN* shall include the descriptor of *C*.

3) *C* shall not be referenced in the <query expression> included in any view descriptor or the <search condition> included in any constraint descriptor.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the collating sequence identified by *C*.

General Rules

1) For every collation descriptor *CD* that includes *CN*, *CD* is modified such that it does not include *CN*. If *CD* does not include any translation name, then *CD* is modified to indicate that it utilizes the DEFAULT collation for its character repertoire.

2) For every character set descriptor *CSD* that includes *CN*, *CSD* is modified such that it does not include *CN*. If *CSD* does not include any translation name, then *CSD* is modified to indicate that it utilizes the DEFAULT collation for its character repertoire.

3) For every column descriptor or domain descriptor *DD* that includes *CN*, *DD* is modified such that it does not include *CN*.

   Note: This causes the column or domain described by *DD* to revert to the default collation for its character set.

4) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE USAGE ON COLLATION *CN* FROM *A* CASCADE

5) The descriptor of *C* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <drop behavior> shall not specify CASCADE.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <drop collation statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.41 <translation definition>

Function

Define a character translation.

Format

```
<translation definition> ::=
       CREATE TRANSLATION <translation name>
         FOR <source character set specification>
           TO <target character set specification>
         FROM <translation source>

<source character set specification> ::= <character set specification>

<target character set specification> ::= <character set specification>

<translation source> ::=
         <translation specification>
       | <translation routine>

<translation specification> ::=
         <external translation>
       | IDENTITY
       | <schema translation name>
       | <internal translation source>

<external translation> ::=
       EXTERNAL <left paren> <quote> <external translation name> <quote> <right paren>

<external translation name> ::=
         <standard translation name>
       | <implementation-defined translation name>

<standard translation name> ::= <translation name>

<implementation-defined translation name> ::= <translation name>

<schema translation name> ::= <translation name>

<translation routine> ::=
         ROUTINE <left paren> <implementation-defined routine name>
             <left paren> <params> <right paren> <right paren>

<params> ::= !! Not yet defined

<internal translation source> ::=
       <left paren> <translation options> <right paren>

<translation options> ::=
       <translation option> [ { <comma> <translation option> }... ]

<translation option> ::=
         USING <left paren> <translation basis> <right paren>
       | MODIFY <left paren> <translation modifiers> <right paren>

<translation modifiers> ::=
```

```
        <translation modifier> [ { <comma> <translation modifier> }... ]

<translation modifier> ::=
      <collating chars> <equals operator> <collating chars>

<translation basis> ::=
      <translation definition> [ { <asterisk> <translation definition> }... ]
```

Note that the BNF nonterminal "<params>" is not defined.

Syntax Rules

1) If a <translation definition> is contained in a <schema definition> and if the <translation name> immediately contained in the <translation definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <translation name> shall not include a translation descriptor whose translation name is <translation name>.

3) A <standard translation name> shall be the name of a translation defined by a national or international standard. An <implementation-defined translation name> shall be the name of a translation that is implementation-defined.

4) The <standard translation name>s and <implementation-defined translation name>s that are supported are implementation-defined. Each translation identified by a <standard translation name> or by an <implementation-defined translation name> shall have associated with it a privilege descriptor that was effectively defined by th <grant statement>

   GRANT USAGE ON TRANSLATION *TRANS* TO PUBLIC

   where *TRANS* is the <standard translation name> or <implementation-defined translation name>.

5) Let *TD* be the translation descriptor identified by <schema translation name>. If the <translation definition> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <schema translation name> shall include *TD*. If the <translation definition> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <schema translation name> shall include *TD* or *S* shall include a <schema element> that creates *TD*.

6) An <implementation-defined routine name> shall be the name of an implementation-defined translation routine.

7) USING shall not be simply contained in <translation options> more than once.

8) In <translation options>, USING shall be specified before MODIFY.

9) In a <translation modifier>, the characters in the first <collating chars> shall be contained in the <target character set specification> and the characters in the second <collating chars> shall be contained in the <source character set specification>.

Access Rules

1) If a <translation definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <translation name>.

2) If <external translation name> is specified, then the applicable privileges shall include USAGE.

General Rules

1) A <translation definition> defines a translation.

2) IDENTITY specifies a translation that makes no changes to the characters.

3) For every character in the second <collating chars>,

   Case:

   a) If the *i*-th character in the first <collating chars> exists, then the *i*-th character in the second <collating chars> is translated into the *i*-th character in the first <collating chars>.

   b) Otherwise, the *i*-th character in the second <collating chars> is not translated.

4) If a <translation basis> specifies more than one <translation definition>, the character string is converted once for each <translation definition>, and the <translation definition>s are applied in the reverse order of their specification.

5) A translation descriptor is created for the defined translation.

6) A privilege descriptor *PD* is created that defines the USAGE privilege on this translation to the <authorization identifier> of the schema or <module> in which the <translation definition> appears. The grantor of the privilege descriptor is set to the special grantor value "_SYSTEM".

7) *PD* is grantable if and only if the USAGE privilege for the <authorization identifier> of the schema or <module> in which the <translation definition> appears is also grantable on every <character set name> contained in the <translation definition>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <translation source> shall not be a <translation routine>.

   b) A <translation specification> shall not be an <internal translation source>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <translation definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.42 <drop translation statement>

Function

Destroy a character translation.

Format

```
<drop translation statement> ::=
      DROP TRANSLATION <translation name>
```

Syntax Rules

1) Let *T* be the translation identified by the <translation name> and let *TN* be the name of *T*.

2) The schema identified by the explicit or implicit schema name of *TN* shall include the descriptor of *T*.

3) *T* shall not be referenced in any <view definition>, the <query expression> included in any view descriptor or in the <search condition> included in any constraint descriptor or be included in any collation descriptor.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the translation identified by *T*.

General Rules

1) Let *CSD* be any <character set definition> that references *T*. *CSD* is modified by deleting any occurrences of a <translation collation> that contains *TN*.

2) Let *A* be the current <authorization identifier>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE USAGE ON TRANSLATION *TN* FROM *A* CASCADE

3) The descriptor of *T* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <drop translation statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.43 <assertion definition>

Function

Specify an integrity constraint by means of an assertion and specify when the assertion is to be checked.

Format

```
<assertion definition> ::=
       CREATE ASSERTION <constraint name>
         <assertion trigger>...
         <triggered assertion> [ <constraint attributes> ]

<triggered assertion> ::=
       CHECK <left paren> <search condition> <right paren>
         [ FOR [ EACH [ ROW OF ] ] <table name> ]

<assertion trigger> ::=
         <immediate assertion trigger>
       | <deferred assertion trigger>

<deferred assertion trigger> ::= BEFORE COMMIT

<immediate assertion trigger> ::=
       AFTER { <assertion trigger statement>
         [ { <comma> <assertion trigger statement> }... ] ON <table name> }...

<assertion trigger statement> ::=
         INSERT
       | DELETE
       | UPDATE [ OF <left paren> <assertion column list> <right paren> ]

<assertion column list> ::= <column name list>
```

Syntax Rules

1) If an <assertion definition> is contained in a <schema definition> and if the <constraint name> contains a <schema name>, then that <schema name> shall be the same as the explicit or implicit <schema name> of the containing <schema definition>.

2) The schema identified by the explicit or implicit schema name of the <constraint name> shall not include a constraint descriptor whose constraint name is <constraint name>.

3) If <constraint attributes> is not specified, then INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

4) The <search condition> shall not contain a <parameter specification>.

5) No <query expression> in the <search condition> shall reference a temporary view or a temporary table.

6) The <search condition> shall not generally contain a <datetime value function> or a <value specification> that is CURRENT_USER, SESSION_USER, or SYSTEM_USER.

7) The <qualified identifier> of <constraint name> shall be different from the <qualified identifier> of the <constraint name> of any other constraint defined in the same schema.

8) The <search condition> shall not generally contain a <query specification> or a <query expression> that is possibly non-deterministic.

9) Let *T1* be the <table name> referenced in the <immediate assertion trigger>. *T1* shall be defined in the containing schema.

10) If <assertion column list> is specified, then each <column name> in the <assertion column list> shall identify a column of *T1*.

11) If FOR is specified, then:

    a) Let *T2* be the <table name> identified in the <triggered assertion>.

    b) *T2* shall be defined in the containing schema.

    c) The <search condition> shall not contain a <set function specification> that is not contained in a <subquery>.

    d) The scope of *T2* is the entire <triggered assertion>.

Access Rules

1) If an <assertion definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <constraint name> of the <assertion definition>.

2) Let *T* be any table referenced in the <search condition> of the <assertion definition>. If *T* is a base table or a view, then

   Case:

   a) For each column *C* of *T* whose <column name> is contained in the <search condition>, the applicable privileges shall include REFERENCES for *C*.

   b) Otherwise, the applicable privileges shall include REFERENCES for at least one column of *T*.

General Rules

1) An <assertion definition> defines an assertion.

   Note: Subclause 10.7, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) Let *SC* be the <search condition> simply contained in the <assertion definition>.

3) The assertion is not satisfied if and only if the result of evaluating *SC* is false.

4) An assertion descriptor is created that describes the assertion being defined. The name included in the assertion descriptor is <constraint name>.

   The assertion descriptor includes an indication of whether the constraint is deferrable or not deferrable and whether the initial constraint mode is *deferred* or *immediate*.

   The assertion descriptor includes *SC*.

5) If the character representation of *SC* cannot be represented in the Information Schema without truncation, then a completion condition is raised: *warning—search condition too long for information schema*.

6) If an <assertion definition> specifies an <immediate assertion trigger>, then

   Case:

   a) If INSERT is specified, the evaluation of <triggered assertion>s takes place at the end of an <insert statement> whose subject table is *T1*.

   b) If DELETE is specified, the evaluation of <triggered assertion>s takes place at the end of each <delete statement: positioned> or <delete statement: searched> whose subject table is *T1*.

   c) If UPDATE is specified without an <assertion column list>, the evaluation of <triggered assertion>s takes place at the end of each <update statement: positioned> or <update statement: searched> whose subject table is *T1*.

   d) If UPDATE is specified with an <assertion column list>, the evaluation of <triggered assertion>s takes place at the end of each <update statement: positioned> or <update statement: searched> whose subject table is *T1* and whose <set clause list> identifies any of the column names in the <assertion column list>.

7) An SQL-statement that triggers an <immediate assertion trigger> is referred to as the triggering statement for the assertion.

8) If <deferred assertion trigger> is specified, then the evaluation of <triggered assertion>s takes place immediately before the execution of a <commit statement>.

9) Case:

   a) If FOR is specified, the <triggered assertion> is evaluated for each row of *T2*. Any outer reference to a column of *T2* is considered to be a reference to the value of the column in that row of *T2*.

      Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

   b) If FOR is not specified, then the <triggered assertion> is evaluated once.

10) The inclusion of EACH ROW OF has no effect on the evaluation of the assertion.

11) If the constraint mode of the assertion in an SQL-transaction is *deferred*, then the checking of an assertion triggered by an <immediate assertion trigger> is deferred.

12) If *SC* causes some column *CN* be to known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of *CN* is changed to known not nullable.

    Note: The nullability characteristic of a column is defined in Subclause 4.19, "Columns, fields, and attributes".

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <assertion definition> shall contain no <assertion trigger>.

   b) A <triggered assertion> shall not specify FOR.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.44 <drop assertion statement>

Function

Destroy an assertion.

Format

```
<drop assertion statement> ::=
      DROP ASSERTION <constraint name>
```

Syntax Rules

1) Let *A* be the assertion identified by <constraint name> and let *AN* be the name of *A*.

2) The schema identified by the explicit or implicit schema name of *AN* shall include the descriptor of *A*.

Access Rules

1) The current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of the assertion identified by *AN*.

General Rules

1) Let *SC* be the <search condition> included in the descriptor of *A*. If *SC* causes some column *CN* be to known not nullable and no other constraint causes *CN* to be known not nullable, then the nullability characteristic of *CN* is changed to possibly nullable.

   Note: The nullability characteristic of a column is defined in Subclause 4.19, "Columns, fields, and attributes".

2) The descriptor of *A* is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <drop assertion statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.45 <trigger definition>

Function

Define triggered SQL-statements.

Format

```
<trigger definition> ::=
```

ANSI Only--SQL3

```
       [ CREATE ] TRIGGER [ <trigger name> ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
       CREATE TRIGGER <trigger name>
```

```
         <trigger action time> <trigger event>
```

ANSI Only--SQL3

```
           [ ON <table name> ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
           ON <table name>
```

```
           [ ORDER <order value> ]
           [ REFERENCING <old or new values alias list> ]
         <triggered action>

<order value> ::= <unsigned integer>

<trigger action time> ::=
         BEFORE
       | AFTER
       | INSTEAD OF

<trigger event> ::=
         INSERT
       | DELETE
       | UPDATE [ OF <trigger column list> ]

<trigger column list> ::= <column name list>

<triggered action> ::=
       [ FOR EACH { ROW | STATEMENT } ]
         [ WHEN <left paren> <search condition> <right paren> ]
         <triggered SQL statement>

<triggered SQL statement> ::=
         <SQL procedure statement>
       | BEGIN
```

```
          { <SQL procedure statement> <semicolon> }...
        END

<old or new values alias list> ::=
      <old or new values alias>...

<old or new values alias> ::=
        OLD [ AS ] <old values correlation name>
      | NEW [ AS ] <new values correlation name>
      | OLD_TABLE [ AS ] <old values table alias>
      | NEW_TABLE [ AS ] <new values table alias>

<old values table alias> ::= <identifier>

<new values table alias> ::= <identifier>

<old values correlation name> ::= <correlation name>

<new values correlation name> ::= <correlation name>
```

Syntax Rules

1) Case:

   a) If a <trigger definition> is contained in a <schema definition> and if the <trigger name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If a <trigger definition> is contained in a <module> and if the <trigger name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the <module>.

   ANSI Only—caused by ISO changes not yet considered by ANSI

2) Case:

   a) If the <trigger definition> is contained in a <query expression>, then none of CREATE, <trigger name>, ON <table name>, or ORDER shall be specified. The <trigger definition> is associated with the table *T* specified by the <query expression>, which is the *subject table* of the <trigger definition>.

   b) Otherwise, CREATE, <trigger name>, and ON <table name> shall all be specified. The table *T* identified by the <table name> *TN* is the *subject table* of the <trigger definition>.

****Editor's Note****

The previous Rule makes it permissible to define a trigger on a <query expression>, which defines an unnamed virtual table. However, succeeding Rules require that the table on which the trigger is defined have a name. This has been identified as a Possible Problem. See Possible Problem **269** in the Editor's Notes.

3) Case:

   a) If a <trigger column list> is specified, then:

      i) No <column name> shall appear more than once in the <trigger column list>.

      ii) The <column name>s of the <trigger column list> shall identify columns of *T*.

   b) Otherwise, there is an implied <trigger column list> that identifies all the columns of *T*.

4) The <triggered SQL statement> shall not generally contain an <SQL transaction statement>, an <SQL connection statement>, or an <SQL session statement>.

ISO Only–SQL3

5) Let *T* be the <table name> of a <trigger definition>. The table identified by *T* is referred to as the *subject table* of the <trigger definition>.

6) If a <trigger column list> is specified, then the <column name>s of the <trigger column list> shall identify columns of the table identified by *T*. Otherwise there is an implied <trigger column list> that identifies all the columns defined in the <table definition> of *T*.

7) The <triggered SQL statement> shall not generally contain an <SQL transaction statement>, an <SQL connection statement>, or an <SQL session statement>.

8) No <column name> shall appear more than once in the <triggered column list>.

9) If REFERENCING is specified, then:

   a) OLD, NEW, OLD_TABLE, and NEW_TABLE shall be specified at most once each within the <old or new values alias list>.

   b) Case:

      i) If <trigger event> specified INSERT and <trigger action time> specifies BEFORE, then neither OLD nor OLD_TABLE shall be specified.

      ii) If <trigger event> specified DELETE and <trigger action time> specifies AFTER, then neither NEW nor NEW_TABLE shall be specified.

   c) No two of <old values correlation name>, <old values table alias>, <new values correlation name>, <new values table alias>, and
      ANSI *TN*,
      ISO *T*,
      treated as a <character string literal>, shall compare equal.

ISO Only–SQL3

d) Let *TN* be the <table name> specified in the <delete statement: searched>, <update statement: searched>, or <insert statement> that is a <triggered SQL statement>. No two of <old values correlation name>, <old values table alias>, <new values correlation name>, <new values table alias>, and *TN*, treated as a <character string literal>, shall compare equal.

ANSI Only—caused by ISO changes not yet considered by ANSI

e) No <table name> contained in the <triggered SQL satement> shall be equal, when treated as a <character string literal>, to <old values correlation name>, <old values table alias>, <new values correlation name>, or <new values table alias>.

f) The scope of the <old values correlation name>, <old values table alias>, <new values correlation name>, <new values table alias>, and
ANSI *TN*
ISO *T*
is the entire <trigger definition>.

10) The <search condition> and <triggered SQL statement>s shall not contain a <value specification> that specifies a parameter reference.

11) If neither FOR EACH ROW nor FOR EACH STATEMENT is specified, then FOR EACH STATEMENT is implicit.

12) If <old values table alias> or <new values table alias> is specified, then:

a) FOR EACH STATEMENT shall be specified.

b) The <identifier> contained in <old values table alias> or <new values table alias> shall be referenced only in a <triggered action> containing FOR EACH STATEMENT.

13) If an <old values table alias> or <new values table alias> is referenced in a <triggered action>, then it is treated as a reference to a table that is not inherently updatable.

14) If <old values correlation name> or <new values correlation name> is specified, then:

a) FOR EACH ROW shall be specified.

b) The <identifier> contained in <old values correlation name> or <new values correlation name> shall be referenced only in a <triggered action> containing FOR EACH ROW.

15) If a <triggered action> contains an outer reference to *T*, then the <trigger definition> shall specify FOR EACH ROW.

16) If a <trigger action time> of INSTEAD OF is specified, then any <triggered action> containing an outer reference to *T* shall contain an <item qualifier>.

Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

17) If BEFORE INSERT or AFTER DELETE is specified and a REFERENCES clause is not specified, then the <triggered action> shall not specify an outer reference to *T*.

18) If the <trigger event> specifies DELETE, then *T* shall not be

Case:

a) the referencing table in any <referential constraint definition> that specifies ON DELETE CASCADE.

b) the referenced table in any <referential constraint definition> that specifies PENDANT.

19) If the <trigger event> specifies UPDATE, then let C_i be the *i*-th <column name> in the <trigger column list>. *T* shall not be the referencing table in any <referential constraint definition> that specifies ON UPDATE CASCADE, ON UPDATE SET NULL, ON UPDATE SET DEFAULT, ON DELETE SET NULL, or ON DELETE SET DEFAULT and contains a <reference column list> that includes C_i.

20) The schema identified by the explicit or implicit schema name of
ANSI *TN*
ISO the <trigger name>
shall not include a trigger descriptor whose trigger name is
ANSI *TN*.
ISO <trigger name>.

ANSI Only—caused by ISO changes not yet considered by ANSI

---

21) Let *D1* and *D2* be two triggers associated with the same subject table and having <trigger action time> INSTEAD OF.

Case:

a) If the <trigger event> of *D1* specifies INSERT or DELETE, then *D2* shall not specify an identical <trigger event>.

b) If the <trigger event> of *D1* specified UPDATE, then *D2* shall not specify a <trigger column list> that references any of the <column name>s in the <trigger column list> of *D1*.

---

22) The *trigger action graph* is a directed graph whose nodes are defined as follows:

a) Let *T* be the
ANSI subject table in any
ISO <table name> that is the subject table name in a
<trigger definition> and let *E* be the <trigger event> in that <trigger definition>. TABLE(*T*) is a node of the graph.

b) If *E* specifies DELETE, then DELETE(*T*) is a node of the graph.

c) If *E* specifies INSERT, then INSERT(*T*) is a node of the graph.

d) If *E* specifies UPDATE, then UPDATE(*T*) and UPDATE(*T*.*C*) are nodes on the graph for each <column name> *C* appearing in the explicit or implicit <trigger column list>.

e) Let T_s be any table that appears as the <table reference> of any <triggered SQL statement> or let T_s be any leaf generally underlying table of any view V_s that appears as the <table reference> of any <triggered SQL statement>.

f) If the <triggered SQL statement> is an <insert statement>, then INSERT(T_s) and TABLE(T_s) are nodes of the graph.

g) If the <triggered SQL statement> is a <delete statement: searched>, then DELETE(T_s) and TABLE(T_s) are nodes of the graph.

h) If the <triggered SQL statement> is an <update statement: searched>, then UPDATE($T_s.C_s$), UPDATE(T_s) and TABLE(T_s) are nodes of the graph for each <column name> C_s of T_s that appears explicitly or implicitly as an <object column> in the <triggered SQL statement>.

23) The arcs of the trigger action graph are defined as follows:

a) If *E* specifies DELETE, then there is an arc from DELETE(*T*) to TABLE(*T*) and

Case:

i) If the <triggered SQL statement> contains an <insert statement>, then there is an arc from DELETE(*T*) to INSERT(T_s) and an arc from INSERT(T_s) to TABLE(T_s).

ii) If the <triggered SQL statement> contains a <delete statement: searched>, then there is an arc from DELETE(*T*) to DELETE(T_s) and an arc from DELETE(T_s) to TABLE(T_s).

iii) If the <triggered SQL statement> contains an <update statement: searched> with an <object column> that references a <column name> C_s of T_s, then there is an arc from DELETE(*T*) to UPDATE($T_s.C_s$), an arc from UPDATE($T_s.C_s$) to UPDATE(T_s), and an arc from UPDATE(T_s) to TABLE(T_s).

b) If *E* specifies INSERT, then there is an arc from INSERT(*T*) to TABLE(*T*) and

Case:

i) If the <triggered SQL statement> contains an <insert statement>, then there is an arc from INSERT(*T*) to INSERT(T_s) and an arc from INSERT(T_s) to TABLE(T_s).

ii) If the <triggered SQL statement> contains a <delete statement: searched>, then there is an arc from INSERT(*T*) to DELETE(T_s) and an arc from DELETE(T_s) to TABLE(T_s).

iii) If the <triggered SQL statement> contains an <update statement: searched> with an <object column> that references a <column name> C_s of T_s, then there is an arc from INSERT(*T*) to UPDATE($T_s.C_s$), an arc from UPDATE($T_s.C_s$) to UPDATE(T_s) and an arc from UPDATE(T_s) to TABLE(T_s).

c) If *E* specifies UPDATE and references a <column name> *C* in a <trigger column list> either explicitly or implicitly, then there is an arc from UPDATE(*T.C*) to UPDATE(*T*), an arc from UPDATE(*T*) to TABLE(*T*) and

Case:

i) If the <triggered SQL statement> contains an <insert statement>, then there is an arc from UPDATE(*T*) to INSERT(T_s) and an arc from INSERT(T_s) to TABLE(T_s).

ii) If the <triggered SQL statement> contains a <delete statement: searched>, then there is an arc from UPDATE(*T*) to DELETE(T_s) and an arc from DELETE(T_s) to TABLE(T_s).

iii) If the <triggered SQL statement> contains an <update statement: searched> with an <object column> that references a <column name> C_s of T_s, then there is an arc from UPDATE(T) to UPDATE($T_s.C_s$), an arc from UPDATE($T_s.C_s$) to UPDATE(T_s), and an arc from UPDATE(T_s) to TABLE(T_s).

24) Let *G* be the trigger action graph and let *T* be the <table name> of the subject table in a <trigger definition> *D*. The *TD_trigger definition* subgraph of *G* is defined as follows:

a) Case:

i) If the <trigger event> of *D* specifies UPDATE, then for any column *C* of table *T*, if UPDATE(*T*.*C*) is a node of *G*, then that node is a node of the subgraph.

ii) If the <trigger event> of *D* specifies DELETE, then DELETE(*T*) is a node of the subgraph.

iii) If the <trigger event> of *D* specifies INSERT, then INSERT(*T*) is a node of the subgraph.

b) The following rule is applied recursively:

If *N* is a node of the subgraph and *N1* is a node of *G* and there exists an arc from *N* to *N1* in *G*, then the node denoted by *N1* is a node of the subgraph and the arc from *N* to *N1* is an arc of the subgraph.

c) A TD_trigger definition subgraph of *G* is defined to be *consistent* if and only if for each node in the subgraph other than those of the form UPDATE(*X*) for some table *X*, the node has at most 1 incoming arc.

25) The TD_trigger definition subgraph of the trigger action graph shall be consistent.

Access Rules

1) If a <trigger definition> is contained in a <module>, then the <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <trigger name> of the <trigger definition>.

2) Let *TN* be any <table name> generally contained in a <search condition> contained in the <trigger definition> or generally contained in a <subquery> in a <triggered SQL statement> contained in the <trigger definition>. The applicable privileges for *TN* shall include SELECT.

3) Let *T* be the subject table of the <trigger definition>. The applicable privileges for *T* shall include TRIGGER.

4) Let *RT* be the table identified by the <table reference> of a <triggered SQL statement> *S*. The applicable privileges for *RT* shall include:

Case:

a) If *S* is an <insert statement>, then

Case:

i) If *S* contains an <insert column list>: INSERT for each <column name> in the <insert column list>.

ii) Otherwise: INSERT.

b) If *S* is an <update statement: searched>: UPDATE for each <object column>.

c) If *S* is a <delete statement: searched>: DELETE.

****Editor's Note****
The preceding Rule is required (by ANSI-only changes?) to check all DML statements that might be executed, including those in invoked routines. This has been identified as a possible problem (see Possible Problem **414** in the Editors Notes).

General Rules

1) A <trigger definition> defines a trigger.

2) A trigger descriptor is created
ANSI for <trigger definition>s not contained in <query expression>s
as follows:

a) The trigger name included in the trigger descriptor is <trigger name>.

b) The subject table included in the trigger descriptor is <table name>.

c) The trigger action time included in the trigger descriptor is <trigger action time>.

d) The trigger event included in the trigger descriptor is <trigger event>.

e) If the <trigger definition> specifies an <old values correlation name>, then the old values correlation name included in the trigger descriptor is that <old values correlation name>.

f) If the <trigger definition> specifies an <new values correlation name>, then the new values correlation name included in the trigger descriptor is that <new values correlation name>.

g) The trigger action included in the trigger descriptor is the specified <triggered action>.

h) If the <trigger event> specifies UPDATE, then the trigger column list included in the trigger descriptor is the implicit or explicit <trigger column list>.

i) The action order included in the trigger descriptor is <order value>.

- 1 Rule deleted.

3) Let

ISO Only–SQL3

VN be the <table name> specified in a <view definition> in which *T* is a leaf generally underlying table. Let *S*, a triggering SQL-statement, be any <SQL data change statement> that specifies a <table reference> of *T* or *VN*. If *S* specifies *T*, then let *RS* be the empty or non-empty set of rows from *T* identified by *S*. If *S* specifies *VN*, then let *RS* be the empty or non-empty set of rows from *T* affected by *S*.

A <trigger definition> specifies the <triggered action> that is effectively

ANSI Only—caused by ISO changes not yet considered by ANSI

S, a triggering SQL statement, be any <SQL data change statement> whose subject table is either a base table *T* or a derived table *T* with (if there is no trigger associated with *T* and *S*

whose <trigger action time> is INSTEAD OF) one or more leaf generally underlying tables *T* that are affected by *S*. For each *T*, let *RS* be the empty or non-empty set of rows affected by *S*.

<trigger definition>s whose subject tables are any *T* specify the <trigger action>s that will be executed whenever *S* is executed.

Case:

a) If BEFORE INSERT is specified, then the <triggered action> is executed before the insert of *RS* into *T*. The rows *RS* are not visible to the <search condition> or <triggered SQL statement>s.

b) If AFTER INSERT is specified, then the <triggered action> is executed after the insert of *RS* into *T*. The rows *RS* are visible to the <search condition> or <triggered SQL statement>s.

c) If INSTEAD OF INSERT is specified, then the triggered action is executed instead of the insertion of *RS* into *T*. If <new values correlation name> or <new values table alias> is specified, then the rows *RS* are visible to the <triggered SQL statement>s.

d) If BEFORE DELETE is specified, then the <triggered action> is executed before the deletion of *RS* from *T*. The rows *RS* are visible to the <search condition> or <triggered SQL statement>s.

e) If AFTER DELETE is specified, then the <triggered action> is executed after the deletion of *RS* from *T*. The rows *RS* are not visible to the <search condition> or <triggered SQL statement>s.

f) If INSTEAD OF DELETE is specified, then the triggered action is executed instead of the deletion of *RS* from *T*. If <old values correlation name> or <old values table alias> is specified, then the rows *RS* are visible to the <triggered SQL statement>s.

g) If BEFORE UPDATE is specified, then the <triggered action> is effectively executed before the update of *RS* in *T*. The values in *RS* that existed prior to the execution of *S* are visible to the <search condition> or <triggered SQL statement>s. If <new values correlation name> is not specified, then the updated values in *RS* are not visible to the <search condition> or <triggered SQL statement>s. If <new values correlation name> is specified, then the updated values in *RS* are accessible to the <search condition> or <triggered SQL statement>s by means of an outer reference that is qualified by the <new values correlation name>.

h) If AFTER UPDATE is specified, then the <triggered action> is effectively executed after the update of *RS* in *T*. The updated values in *RS* are visible to the <search condition> or <triggered SQL statement>s. If <old values correlation name> is not specified, then the values in *RS* that existed prior to the execution of *S* are not visible to the <search condition> or <triggered SQL statement>s. If <old values correlation name> is specified, then the values in *RS* that existed prior to the execution of *S* are accessible to the <search condition> or <triggered SQL statement>s by means of an outer reference that is qualified by the <old values correlation name>.

i) If INSTEAD OF UPDATE is specified, then the triggered action is executed instead of the update of *RS* in *T*. If <new values correlation name> or <new values table alias> is specified, then the rows *RS* are visible to the <triggered SQL statement>s. If <old values correlation name> or <old values table alias> is specified, then the values in *RS* that existed prior to the execution of *S* are visible to the <triggered SQL statement>s.

4) If
ANSI triggers exist for the subject table
ISO trigger descriptors exist
with both FOR EACH STATEMENT and FOR EACH ROW for the same <trigger action time> and the same <trigger event>, then for each
ANSI such trigger
ISO trigger described by such a trigger descriptor

Case:

a) If the <trigger action time> specified is BEFORE or INSTEAD OF, then the execution of the trigger
ISO identified by the trigger descriptor
with the FOR EACH STATEMENT precedes the execution of the trigger
ISO identified by the trigger descriptor
with the FOR EACH ROW.

b) If the <trigger action time> specified is AFTER, then the execution of the trigger
ANSI identified by the trigger descriptor
with the FOR EACH STATEMENT follows the execution of the trigger
ISO identified by the trigger descriptor
with the FOR EACH ROW.

ISO Only—caused by ANSI changes not yet considered by ISO

5) Let *I* be the set of integers contained in the action order of the trigger descriptors having the same event object catalog, event object table, event manipulation, condition timing, and action orientation as the trigger being defined.

Case:

a) If the <trigger definition> specifies ORDER, then let *V* be the value specified by the <order value>. If *V* is equal to the value of any integer in *I*, then the value of action order in all descriptors asociated with *I* having an order value greater than *V* are incremented by one.

b) If the <trigger definition> does not specify ORDER, then

Case:

i) If *I* is empty, then let *V* be 1.

ii) Otherwise, let *V* be one greater than the largest integer in *I*.

6) If at the time of execution of *S*, two or more trigger descriptors are associated with *I*, then the order of execution of the triggers identified by those trigger descriptors is ascending by value of the action order.

ANSI Only—caused by ISO changes not yet considered by ANSI

7) If <trigger definition> is not contained in a <query expression>, then:

   a) Let *I* be the set of integers that are the values of the <order value> of the teirggers for the subject table that have the same <trigger action time> and <trigger event>, and the same choice of FOR EACH STATEMENT or FOR EACH ROW, as the trigger being defined.

   b) Case:

      i) If the <trigger definition> specifies ORDER, then let *V* be the value specified by the <order value>. If *V* is equal to the value of any integer in *I*, then the value of action order in all descriptors associated with *I* having an order value greater than *V* are incremented by one.

      ii) If the <trigger definition> does not specify ORDER, then

      Case:

      1) If *I* is empty, then let *V* be 1.

      2) Otherwise, let *V* be one greater than the largest integer in *I*.

8) If at the time of execution of *S*, two or more triggers for the subject table have the same <trigger action time> and <trigger event>, and the same choice of FOR EACH STATEMENT or FOR EACH ROW, then

   Case:

   a) If <trigger definition> is not contained in a <query expression>, then the order of execution of the triggers is ascending by value of the action order in their trigger descriptors.

   b) Otherwise, the order of execution of the triggers is the order in which they are specified in the <query expression>.

---

9) If a <triggered action> specifies FOR EACH ROW, then the <triggered action> is executed once for each row of *RS*. Otherwise the <triggered action> is executed once.

ANSI Only–SQL3

---

10) If a <triggered action> specifies FOR EACH ROW and that <triggered action> generally contains an <update statement: searched>, then:

    a) Changes caused by execution of the <triggered action> for a row of *RS* are not visible to the <search condition> of that <triggered action> or to the <search condition> of that <update statement: searched> for execution of the <triggered action> for other rows of *RS*.

    b) Changes caused by execution of the <triggered action> for a row of *RS* are visible to the <set clause list> of that <update statement: searched> for execution of the <triggered action> for other rows of *RS*.

---

11) Let *C* be a <column name> of *T* referenced as an outer reference in a <trigger definition>.
    Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

12) If the outer reference to *C* is contained in a <trigger definition> containing a <trigger event> that specifies UPDATE, then let *SU* be a triggering SQL-statement that is an <update statement: positioned> or <update statement: searched> that references *T*.

Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

Case:

a) If the reference to *C* is qualified by <new values correlation name>, then the reference to *C* is to the value that will be present in the column identified by *C* in some row of *RS* after the update of that row by *SU*.

b) If the reference to *C* is qualified by <old values correlation name>, then the reference to *C* is to the value that was present in the column identified by *C* in some row of *RS* before the update of that row by *SU*.

c) If the reference to *C* is not qualified by either <new values correlation name> or <old values correlation name>, then

Case:

i) If the <trigger action time> specifies AFTER, then the reference to *C* is to the value that will be present in the column identified by *C* in some row of *RS* after the update of that row by *SU*.

ii) If the <trigger action time> specifies BEFORE, then the reference to *C* is to the value that was present in the column identified by *C* in some row of *RS* before the update of that row by *SU*.

13) If the outer reference to *C* is contained in a <trigger definition> containing a <trigger event> that specifies DELETE or INSERT, then the reference to *C* is to the value of *C* in some row of *RS*.

Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

14) If a reference to an <old values table alias> is contained in a <triggered action>, then it refers to the whole collection of rows *RS* before any effect of the triggering SQL-statement.

15) If a reference to a <new values table alias> is contained in a <triggered action>, then:

Case:

a) If the <trigger event> is INSERT, then the reference is to the whole collection of candidate rows *RS* to be inserted, that is, with defaults already applied.

b) If the <trigger event> is UPDATE, then the reference is to the whole collection of rows *RS* that would have resulted from updating the original collection *RS* as though it were a base table.

16) Case:

a) If a <triggered action> contains a <search condition>, then

Case:

i) If the <search condition> is true, then the <triggered SQL statement>s are executed in the order in which they are specified.

ii) If the <search condition> is not true, then the <triggered SQL statement>s are not executed.

b) If a <triggered action> does not contain a <search condition>, then the <triggered SQL statement>s are executed in the order in which they are specified.

17) If a <triggered SQL statement> contains more than one <SQL procedure statement>, then the <SQL procedure statement>s are effectively executed in the order in which they are specified in the <triggered SQL statement>.

ISO Only–SQL3

18) Let *ID* be the <table name> that is the subject of any <triggered SQL statement> that is an <insert statement>. Let *DT* be the <table name> that is the subject of any <triggered SQL statement> that is a <delete statement: searched>. Let *UT* be the <table name> that is the subject of any <triggered SQL statement> that is an <update statement: searched>. A <triggered action> is *cascaded* when

Case:

a) *ID* appears as the <table name> in a <trigger definition> that has a <trigger event> that specifies INSERT.

b) *DT* appears as the <table name> in a <trigger definition> that has a <trigger event> that specifies DELETE.

c) *UT* appears as the <table name> in a <trigger definition> that has a <trigger event> that specifies UPDATE.

19) If the execution of a <triggered SQL statement> inserts rows, deletes rows, or updates rows of a table *TX*, then any cascaded triggered actions for *TX* are activated by the <triggered SQL statement>s as defined by the <trigger definition>s containing the <table name> *TX*.

ANSI Only—caused by ISO changes not yet considered by ANSI

20) A <triggered action> *TA* is *cascaded* if:

a) A <triggered SQL statement> causes execution of any <SQL data change statement> *SDCS* whose immediately contained <table reference> or <table name> identifies a table *TX*, and

b) *TA* is the <triggered action> of a trigger *TR* whose subject table is *TX*, and

c) Case:

i) If *SDCS* is an <insert statement>, then the <trigger event> of *TR* is INSERT.

ii) If *SDCS* is a <delete statement: positioned> or <delete statement: searched>, then the <trigger event> of *TR* is INSERT.

iii) If *SDCS* is an <update statement: positioned> or <update statement: searched>, then the <trigger event> of *TR* is UPDATE.

21) If the execution of a <triggered SQL statement> inserts rows into, deletes rows from, or updates rows of a table *TX*, then any cascaded <triggered action>s for *TX* are executed.

---

22) If the execution of the <triggered SQL statement> for two rows R_1 and R_2 of *RS* result in the update of a column *C* for the same row R_x in some table and if the <triggered SQL statement> for R_1 and R_2 result in different values for *C*, then an exception condition is raised: *triggered data change violation*.

ISO Only–SQL3

---

23) Let *TN* be the trigger name of *T*. Let *S* be the schema identified by the <schema name> explicitly or implicitly contained in *TN*. The current authorization identifier during the execution of the <triggered SQL statement>s of *T* is the <authorization identifier> of the owner of *S*.

ANSI Only—caused by ISO changes not yet considered by ANSI

---

24) If <trigger definition> is not contained in a <query expression>, then let *TN* be the trigger name of *T*. Let *S* be the schema identified by the <schema name> explicitly or implicitly contained in *TN*. The current authorization identifier during the execution of the <triggered SQL statement> of *T* is the <authorization identifier> of the owner of *S*.

---

25) If the cursor mode of the current SQL-transaction is set to *cascade off*, then all <triggered SQL statement>s are effectively deferred until implicitly enacted by execution of a <commit statement> or a <close statement>.

26) If the execution of a <triggered SQL statement> is not successful, then an exception condition is raised: *triggered action exception*. The exception information associated with the <triggered SQL statement> is entered into the diagnostics area in a location other than the location corresponding to condition number 1.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <trigger definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.46 <drop trigger statement>

Function

Destroy a trigger.

Format

```
<drop trigger statement> ::= DROP TRIGGER <trigger name>
```

Syntax Rules

1) Let *TR* be the trigger identified by the <trigger name>.

Access Rules

1) The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the <schema name> of *TR*.

General Rules

- 1 Rule deleted.

1) The identified trigger and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop trigger statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

- 1 Subclause moved to Part 4

11.47 <abstract data type definition>

Function

Define an abstract data type.

Format

```
<abstract data type definition> ::= CREATE <abstract data type body>
```

Syntax Rules

1) Case:

   a) If an <abstract data type definition> is contained in a <schema definition>, and if the <abstract data type definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If an <abstract data type definition> is contained in a <module>, and if the <abstract data type definition> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <module>.

- 1 Rule moved.

Access Rules

None.

General Rules

1) An <abstract data type definition> defines an abstract data type and the functions that are associated with it.

2) A value of an abstract data type consists of a value of the attributes specified by the ordered sequence of attribute descriptors in the abstract data type descriptor. The values corresponding to the actual base types of such an abstract data type value are the *actual base items* of that value.

3) A privilege descriptor is created that defines the USAGE privilege for the <abstract data type name> to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <abstract data type name>. This privilege is grantable. The grantor for this privilege descriptor is set to the special grantor value "_SYSTEM".

4) A privilege descriptor is created granting the UNDER privilege on the abstract data type to the owner; the grantor is the special grantor value "_SYSTEM". If and only if the UNDER privilege is held on all supertypes of the abstract data type WITH GRANT OPTION will be privilege descriptor be specified WITH GRANT OPTION.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.48 <abstract data type body>

Function

Specify an <abstract data type body>.

Format

```
<abstract data type body> ::=
       { VALUE | OBJECT } TYPE <abstract data type name>
       [ <oid options> ]
       [ <subtype clause> ]
       [ <constructor option> ]
       [ <constructor option> ]
       [ [ <constant or updatable> ] [ <member list> ] ]
       [ <type default> ]

<oid options> ::=
       WITH OID [ [ NOT ] VISIBLE ]
```

- 1 alternative deleted.

```
<subtype clause> ::=
       UNDER <supertype clause> [ { <comma> <supertype clause> }... ]

<supertype clause> ::=
       <abstract data type name> [ <component renaming clause> ]

<component renaming clause> ::=
       WITH <left paren> <component renaming element>
           [ { , <component renaming element> }... ] <right paren>

<component renaming element> ::=
       <supertype component name> AS <subtype component name>

<supertype component name> ::= <component name>

<subtype component name> ::= <component name>

<constructor option> ::=
       CONSTRUCTOR <option level>

<destructor option> ::=
       DESTRUCTOR <option level>

<constructor level> ::=
         PRIVATE
       | PROTECTED
       | PUBLIC

<member list> ::=
       <left paren> <member> [ { <comma> <member> }... ] <right paren>

<member> ::=
         <attribute definition>
       | <routine declaration>
       | <operator name list>
       | <ordering clause>
```

```
      | <cast clause>
```

ANSI Only--SQL3

```
      | <table constraint definition>
```

```
<operator name list> ::= OPERATORS <specific routine designator>...

<ordering clause> ::=
        RELATIVE <relative function specification>
      | HASH <hash function specification>
      | EQUALS <equals function specification>
        [ LESS THAN <less-than function specification> ]

<cast clause> ::=
      CAST <left paren> <operand data type> AS <result data type>
           WITH <cast function> <right paren>

<cast function> ::=
      <specific routine designator>

<equals function specification> ::=
        <specific routine designator>
      | STATE
      | OID

<less-than function specification> ::=
        <specific routine designator>
      | NONE

<relative function specification> ::=
      <specific routine designator>

<hash function specification> ::=
      <specific routine designator>

<routine declaration> ::=
      [ <encapsulation level> ] <routine specification>

<operand data type> ::= <data type>

<result data type> ::= <data type>

<type default> ::=
      TYPE <default clause>
```

Syntax Rules

1) Case:

   a) If <abstract data type body> immediately contains CONSTANT, then CONSTANT is implicit for each <attribute definition> contained in the <member list>.

b) If <abstract data type body> immediately contains UPDATABLE, then there shall exist at least one <attribute definition> contained in the <member list> that does not specify CONSTANT.

2) The <qualified identifier> of <abstract data type name> shall be different from the <qualified identifier> of any other <abstract data type name> of an <abstract data type definition> or <distinct type definition> in the same schema and shall be different from the <qualified identifier> of any <domain name> in the same schema.

3) <operator name list> shall be specified no more than once.

4) Let *STD* be the containing <abstract data type definition> and let *ST* be the abstract data type defined by *STD*.

5) For *ST*, the Syntax Rules of Subclause 8.3, "Type precedence list determination", in Part 4 of this
ANSI American
ISO International
Standard, are applied, yielding the type precedence list *TPL* of *ST*. There shall be a *TPL*.

6) If <subtype clause> is not specified, then <member list> shall be specified.

7) If <subtype clause> is specified, then

Case:

a) If VALUE is specified, then every abstract data type identified by an <abstract data type name> in the <subtype clause> shall be a value abstract data type.

b) If OBJECT is specified, then every abstract data type identified by an <abstract data type name> in the <subtype clause> shall be an object abstract data type.

8) An <oid options> shall be specified only if OBJECT is specified. If OBJECT is specified an <oid options> is not specified, then WITH OID NOT VISIBLE is implicit.

9) If an <abstract data type body> contains a <subtype clause>, then:

a) Each <abstract data type name> simply contained in the <subtype clause> identifies a direct supertype of the abstract data type *ST*.

b) *ST* is a direct subtype of each abstract data type identified by an <abstract data type name> simply contained in the <subtype clause>.

10) If <subtype clause> is specified, then:

a) No <abstract data type name> shall be simply contained more than once in <subtype clause>.

b) Let *m* be the number of direct supertypes of *ST*. Let *k(i)* be the number of attributes whose attribute descriptor is included in the abstract data type descriptor of the direct supertype of *ST* whose abstract data type name is simply contained in the *i* <supertype clause>. Let C_{ij} be the *j*-th attribute descriptor included in the abstract data type descriptor of the *i*-th direct supertype of *ST*.

c) Let the term *replicated attribute* mean an attribute appearing in more than one direct supertype of *ST* that is inherited by at least one of those direct supertypes from a single higher-level supertype.

d) Let S be:

$$C_{11}, \cdots, C_{1k(1)}, \cdots, C_{i1}, \cdots, C_{ik(i)}, \cdots, C_{m1}, \cdots, C_{mk(m)}$$

e) The list S preserves the order of the supertypes ST_i of ST and the order of the attributes in each ST_i.

ISO Only—caused by ANSI changes not yet considered by ISO

f) Attributes whose attribute name is OID are dropped from S.

g) For every replicated attribute SA, all occurrences of SA except the first are dropped from S.

h) S is the inherited representation of ST.

i) Let *inherited attribute* be an attribute in the inherited representation of ST.

j) If <component renaming clause> is specified, then for each <component renaming element>:

i) The <supertype component name> shall identify an attribute or routine defined in or inherited by the abstract data type identified by the <abstract data type name> of the containing <supertype clause>. The <encapsulation level> of that attribute shall be PUBLIC or PROTECTED.

ii) If the <supertype component name> identifies an attribute, then the effective <attribute name> in the inherited representation of STD is the <subtype component name> specified in the <component renaming clause>.

****Editor's Note****

There is a general problem in this section in that where <abstract data type body> is contained in a <type template definition> and not in an <abstract data type definition>, STD is not defined. When the fix to cover type templates is made, it should probably handle this rule by generating merely OID REF NOT NULL. Then <generated type reference> in Subclause 10.3, "<generated type reference>", should tighten REF to REF(*IDN*) whenever it generates a specific type name *IDN* from the template.

ISO Only—caused by ANSI changes not yet considered by ISO

11) No <attribute name> immediately contained in an <attribute definition> explicitly specified in the <abstract data type body> nor any <component name> in a <component renaming element> shall specify OID.

12) Let *ADTN* be the <abstract data type name> of the containing <abstract data type definition>. If <oid options> specifies WITH OID, then an <attribute definition> is implicitly specified that defines an attribute with <attribute name> OID, data type object identifier type with the associated abstract data type identified by *ADTN*, and the <column constraint definition> NOT NULL. Any inherited component with <attribute name> OID is deleted from the inherited representation of STD.

- 1 Rule deleted

13) An <abstract data type definition> *ADT*

ISO Only—caused by ANSI changes not yet considered by ISO

that defines an object ADT is directly based on its own object identifier type (which identifies *ADT* as the associated abstract data type). Otherwise, it

is directly based on a data type *DT* if *DT* is the data type of an <attribute definition> of *ADT*. An <abstract data type definition> *ADT* is based on a data type *DT* if *ADT* is directly based on *DT* or if *ADT* is directly based on an abstract data type *ADT2* that is based on *DT*.

14) The actual base type of a predefined type *PDT* is *PDT*. The actual base types of an abstract data type *ADT* are the predefined
ISO and object identifier
types on which *ADT* is based, and, if they are derived from the components of *ADT*, the actual base types are in the order of the components of *ADT*.

15) An abstract data type shall not be based on itself.

16) If no <ordering clause> is present, then

Case:

a) If VALUE is specified, then an <ordering clause> that specifies EQUALS with an <equals function specification> of STATE is implicit.

b) Otherwise, an <ordering clause> that specifies EQUALS with an <equals function specification> of OID is implicit.

17) If no <less than function specification> is present, then NONE is implicit.

****Editor's Note****
Paper X3H2-93-404R/MUN-101 replaced the rule dealing with the EQUALS function, but did not address the preceding rule dealing with the LESS THAN function. Should this rule also be deleted?

18) At most one <ordering clause> shall be contained in the <member list>.

19) Let *ADTN* be the <abstract data type name>.

Case:

a) If <equals function specification> specifies STATE, then:

i) Let *C1*, . . . , *Cn* be the components of the representation of the abstract data
ANSI type.
ISO type, excluding any component with <attribute name> OID.

ii) The <abstract data type definition> effectively contains an implicit <routine> that specifies:

```
FUNCTION EQUALS ( ADT1 ADTN, ADT2 ADTN )
  RETURNS BOOLEAN
  RETURN
      ( ADT1 .. C1 = ADT2 .. C1 )
    AND ADT1 .. C2 = ADT2 .. C2
       .
       .
       .

    AND ADT1 .. Cn = ADT2 .. Cn )
```

Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

b) If <equals function specification> specifies OID, then:

i) OBJECT shall be specified.

ii) The <abstract data type definition> effectively contains an implicit <routine> that specifies

```
FUNCTION EQUALS ( ADT1 ADTN, ADT2 ADTN )
  RETURNS BOOLEAN
  RETURN
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
      ( ADT1 .. OID = ADT2 .. OID
```

ANSI Only--SQL3

```
      ( OID ADT = OID ADT )
```

Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

20) If a <routine name> is specified as an <equals function specification> or <less-than function specification>, then

Note: <routine name> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

a) Let *FD* be the <routine> identified by the <specific routine designator>. The function referenced by *FD* is called the *equals function* or *less-than function*, respectively.

Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

Note: <specific routine designator> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

b) *FD* shall specify exactly two <operand data type>s, each of which shall specify *TN*.

c) The result data type of *FD* shall be BOOLEAN.

d) If *FD* is an external routine, then *FD* shall specify or imply NOT VARIANT.

Note: In order to provide behavior retaining essential properties of equality, the <routine name> specified as an <equals function specification> should define a predicate that is reflexive ($x=x$), symmetric ($x=y$ if and only if $y=x$), and transitive (if $x=y$ and $y=z$, then $x=z$).

Further, it should be recognized that if $x=y$ in one type, and x and y are cast to another type to become x' and y', then $x'=y'$ may not hold if at least one of the types has overridden the implicit <equals function specification>, or only one of the types is an object ADT with the <equals function specification> OID.

Assignment and equality should work in tandem. Except where behavior such as truncation is exhibited, "SET x = y" should operate such that "NOT x IS DISTINCT FROM y" evaluates to true. This feature of the equality function may interact with any cast functions specified on the ADT.

21) If a <routine name> is specified as a <relative function specification>, then:

Note: <routine name> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

a) Let *FD* be the <routine> identified by the <routine name>. The function referenced by *FD* is called the *relative ordering function*.

Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

b) Let *ADTN* be the <abstract data type name>.

c) *FD* shall specify exactly two <operand data type>s, each of which shall specify *ADTN*.

d) The result data type of *FD* shall be INTEGER.

e) If *FD* is an external routine, then *FD* shall specify or imply NOT VARIANT.

22) If a <routine name> is specified as a <hash function specification>, then:

Note: <routine name> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

a) Let *FD* be the <routine> identified by the <routine name>. The function referenced by *FD* is called the *hash ordering function*.

Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

b) Let *ADTN* be the <abstract data type name>.

c) *FD* shall specify exactly one <operand data type>s, which shall specify *ADTN*.

d) The result data type of *FD* shall be a predefined data type.

e) If *FD* is an external routine, then *FD* shall specify or imply NOT VARIANT.

23) If a <routine declaration> *RD* has no <encapsulation level> specified, then let *ADTB* be the <abstract data type body> that contains *RD*. Let *ML* be the <member list> of *TD*.

Case:

a) If *RD* is the first element of the list *ML* that contains either an <attribute definition> or a <routine declaration>, then <encapsulation level> PUBLIC is implicit.

b) Otherwise, the <encapsulation level> of the nearest preceding element of the list *ML* that contains either an <attribute definition> or a <routine declaration> is implicit.

24) For every <cast clause> *CC*:

a) Let *ADTN* be the <abstract data type name> of the containing <abstract data type definition>. Let *ADT* be the abstract data type.

b) Let *ODT* and *RDT* be the <operand data type> and result data type of *CC*.

c) Exactly one of *ODT* and *RDT* shall be the same as *ADT*.

ANSI Only–SQL3

Let the other be *ADT2*.

d) *ADT2* shall not contain a <cast clause> specifying the same *ODT* and *RDT*.

ISO Only—caused by ANSI changes not yet considered by ISO

e) If both *ODT* and *RDT* are abstract data types, then *ODT* shall be the same as *ADT*.

Note: Thus a <cast clause> specifying CAST(*ADT1* AS *ADT2*) must be specified in *ADT1*. The requirement that this <cast clause> be specified in *ADT1* (rather than *ADT2*) is arbitrary. The essential requirement is that there should be at most one such <cast clause>, so that the CAST rules will be unambiguous.

f) Let *FD* be the <routine> identified by the <specific routine designator> of *CC*. The SQL procedure referenced by *FD* is called the *cast function* of the <cast clause>.

Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

Note: <specific routine designator> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

g) *FD* shall specify exactly one <operand data type>, which shall be the same as *ODT*.

h) The result data type of *FD* shall be the same as *RDT*.

Note: In order to provide behavior for casts that is intuitive, the cast functions ought to be consistent with equality. If *CastT1T2* casts from type *T1* to type *T2*, and the types are in the same type family, and if *CastT1T2*(*x1*) yields *x2*, then it should also be the case that *CastT1T2*(*x1*) = *x2*. Secondly, a cycle of CASTs (a series of cast functions that start and end of the same type), should similarly be consistent with equality, except under conditions such as truncation. Finally, the cast functions defined in a type and its supertype should preserve that hierarchy by casting to another type and its supertype, respectively, rather than casting to a type and its subtype, respectively.

25) If <constructor option> is not specified, then let *CP* be PRIVATE; otherwise, let *CP* be the <option level> immediately contained in <constructor option>.

26) Let *ADTN* be the <abstract data type name> of the <abstract data type definition> that contains <abstract data type body>. The following <routine declaration> is implicit in <abstract data type body> as the last <member> of <member list>:

```
CP FUNCTION ADTN () RETURNS ADTN
   . . .
  RETURN NEWADTNI
```

where *NEWADTNI* is a new instance of the abstract data type identified by
ANSI *ADTN*
ISO *ADTN*, whose OID field (if any) is set to a new unique object identifier, and
whose attributes are uninitialized. This function is the constructor function for the data type identified by *ADTN*.

27) If OBJECT is specified, then:

a) Let *ADTN* be the <abstract data type name> of the <abstract data type definition> that contains the <abstract data type body>.

b) If <destructor option> is specified, then OBJECT shall be specified.

c) If <destructor option> is not specified, then let *DP* be PRIVATE; otherwise, let *DP* be the <option level> immediately contained in <destructor option>.

d) The following <routine declaration> is implicit in <abstract data type body> as a <member> in the <member list>:

DP FUNCTION DESTROY (*TEMP ADTN* RETURNS *ADTN*

This function is the *destructor function* for the abstract data type identified by *ADTN*. The return value is null.

Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

28) The scope of the <attribute name>s and <routine name>s contained in the <member list> of the <abstract data type definition> shall be the entire <abstract data type definition>, including any <routine> contained in the <abstract data type definition>.

Note: <routine> and <routine name> are defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

29) If a <type default> is specified, then the <default option> of the <default clause> shall specify either NULL or a <routine invocation> that identifies a function.

Note: <routine invocation> is defined in Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard.

Access Rules

1) The applicable privileges on each <abstract data type name> specified in <subtable clause> shall include UNDER.

General Rules

1) The <ordering clause> specifies function(s) that define the ordering referenced in the definitions of <predicate>s and the <order by clause>, and the equality reference in the <group by clause>, <having clause>, <unique predicate>, and DISTINCT.

   a) The equals function is implicitly invoked with two arguments *X* and *Y* to obtain a BOOLEAN result that is true if and only if *X* is equal to *Y*. The less-than function is implicitly invoked with two arguments *X* and *Y* to obtain a BOOLEAN result that is true if and only if *X* is less than *Y*.

   b) The relative ordering function is implicitly invoked with two arguments *X* and *Y* to obtain an INTEGER result *Z*. *X* is defined to be less than, equal to, or greater than *Y* if and only if *Z* is less than zero, equal to zero, or greater than zero, respectively.

   c) The hash ordering function is implicitly invoked with an argument *X* if the abstract data type to obtain a result of some predefined data type *PDT*. The relative ordering of *X* in the abstract data type is defined to be the same as the ordering of *PDT* in the predefined data type.

2) The value of an instance of an abstract data type is the ordered sequence of its
ANSI attributes.
ISO attributes, excluding any attribute with attribute name OID.

3) The degree of the abstract data type being created is initially set to *N*, the number of inherited attributes; the General Rules of Subclause 11.7, "<attribute definition>", specify the degree of the abstract data type during the definition of the attributes in that abstract data type.

4) An abstract data type descriptor *STDS* is created that describes the abstract data type being defined. *STDS* include:

   a) The abstract data type name <abstract data type name> simply contained in *STD*;

   b) An indication of whether *ST* is a VALUE ADT or an OBJECT ADT;

   c) The degree of *ST*;

   d) The attribute descriptors of all inherited attributes and their ordinal positions in *ST*;

   e) The attribute descriptors of all simply contained attributes and their ordinal positions in *ST*;

f) The values of any <relative function specification>, <hash function specification>, <equals function specification>, and <less-than function specification>;

g) The descriptors of any <cast function>s;

h) The <abstract data type name>s contained in the <subtype clause>, if present; and

i) The values of all <specific routine designator>s simply contained in the <operator name list>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type body>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.49 <distinct type definition>

Function

Define a distinct abstract data type derived from another abstract data type.

Format

```
<distinct type definition> ::=
      CREATE DISTINCT TYPE <distinct type name> AS <data type>
```

- 1 production deleted.

Syntax Rules

1) Case:

   a) If a < distinct type definition> is contained in a <schema definition> and if the <distinct type name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

   b) If a < distinct type definition> is contained in a <module> and if the <distinct type name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <module>.

2) The <qualified identifier> of <distinct type name> shall be different from the <qualified identifier> of the <distinct type name> of any other distinct type in the same schema, from the <qualified identifier> of the <abstract data type name> of any abstract data type in the same shcmea, and from the <qualified identifier> of the <domain name> of any domain in the same schema.

3) The data type identified by <data type> shall not be a distinct type.

4) A <distinct type definition> is *directly based on* the data type designated by <data type>. The type designated by <data type> is denoted the *source type* of the distinct type defined by <distinct type definition>. A <distinct type definition> *DTD* is *based on* a data type *DT* if *DTD* is directly based on *DT* or if *DTD* is directly based on a data type that is based on *DT*. No <distinct type definition> shall be based on itself.

Access Rules

1) If a <distinct type definition> is contained in a <module>, then the current <authorization identifier> shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <distinct type name>.

2) If <data type> specifies an <abstract data type name> *ADTN*, then the applicable privileges shall include USAGE on the abstract data type identified by *ADTN*.

3) If <data type> specifies a <distinct type name> *DTN*, then the applicable privileges shall include USAGE on the distinct type identified by *DTN*.

General Rules

1) A <distinct type definition> defines distinct type based on the data type identified by <data type>.

2) A distinct type descriptor is created that describes the distinct type being defined:

   a) The distinct type name included in the distinct type desscriptor is <distinct type name>.

   b) The source type name included in the distinct type descriptor is the name of the source type of the distinct type being defined.

   c) An indication of whether the source type of the distinct type being defined is a predefined data type or abstract data type is included.

3) If the source type of the distinct type being defined is an abstract data type, then:

   a) Let *DTN* be the <distinct type name> and *ADTN* be the name of the abstract data type identified by <data type>.

   b) Let *ADTD* be the <abstract data type definition> of the <abstract data type definition> identified by *ADTN* where each occurrence of ADTN is replaced by *DTN*.

   ****Editor's Note****

   In X3H2-93-444/MUN-167, Nelson Mattos commented "I do not understand how this rule can reference <abstract data type definition> since there will be no <abstract data type definition> by the time a distinct type is defined. Shouldn't it reference the descriptor of the corresponding ADT?"

   c) The Syntax and General Rules for *ADTN*, except for rules creating the abstract data type descriptor and privilege descriptors, are applied without further Access Rule checking.

   d) The other values of the distinct type descriptor are those produced by the General Rules of Subclause 11.48, "<abstract data type body>".

   ****Editor's Note****

   In X3H2-93-444/MUN-167, Nelson Mattos commented "Instead of saying that the values in the descriptor are products by the General Rules of Subclause 11.48, "<abstract data type body>", shouldn't we say something like we have in Syntax Rule 15) of Subclause 11.6, "<column definition>"?".

   Ed Dee commented in X3H2-94-242/SOU-054 that there is no such Rule, but he does not know how to update the reference.

4) A privilege descriptor is created that defines the USAGE privilege on the distinct type identified by <distinct type name> to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of <distinct type name>. This privilege is grantable. The grantor for this privilege descriptor is set to the special grantor value "_SYSTEM".

- 1 Rule deleted.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <distinct type definition>.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.50 <type template definition>

Function

Define a type template.

Format

```
<type template definition> ::=
       CREATE TYPE TEMPLATE <type template name>
         <template parameter declaration list>
         <abstract data type body>

<template parameter declaration list> ::=
       <left paren>
         <template parameter declaration>
           [ { <comma> <template parameter declaration> }... ]
       <right paren>

<template parameter declaration> ::=
       <template parameter name> <template parameter type>

<template parameter type> ::=
         <data type>
       | TYPE
```

Syntax Rules

1) If a <type template definition> is contained in a <schema definition>, and if the <type template name> contains a <schema name>, then that <schema name> shall be the same as the specified or implicit <schema name> of the containing <schema definition>.

2) Each <type template definition> for which the <qualified identifier> of <type template name> is the same is said to belong to the same type template family.

3) The sequence of <template parameter type>s in the <template parameter declaration list> shall be different from that of any other <type template definition> in the same type template family.

4) The <template parameter name> of each <template parameter declaration> shall be different from the <template parameter name> of any other <template parameter declaration> in the containing <template parameter declaration list>.

5) The <data type> of a <template parameter type> shall not contain a <data type> that is a <template parameter name>.

6) No <template parameter name> in a <template parameter declaration> shall specify :GEN_TYPE. :GEN_TYPE is a reserved <template parameter name> whose <template parameter type> is TYPE.

7) A <type template definition> *TTD* is directly based on a <type template definition> *TTD2* if *TTD* contains a <generated type reference> that references *TTD2*. A <type template definition> *TTD* is based on a <type template definition> *TTD2* if *TTD* is directly based on *TTD2* or if *TTD* is directly based on a <type template definition> that is based on *TTD2*. No <type template definition> shall be based on itself.

Access Rules

1) If a <type template definition> is contained in a <module>, then the <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <type template definition>.

General Rules

1) A <type template definition> defines a type template.

2) A type template descriptor is created that describes the type template *T* defined by the <type template definition>. The type template descriptor includes the <type template name>, <template parameter declaration list>, and <abstract data type body>.

3) Let *A* be the <authorization identifier> that owns *T*.

4) A privilege descriptor is created that specifies the privilege USAGE on this type template to *A*. This privilege is grantable if and only if all the applicable privileges required to define *T* are grantable. The grantor of this privilege descriptor is set to the special grantor value "_SYSTEM".

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <type template definition>.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.51 <drop type template statement>

Function

Drop a type template and all types generated by the type template.

Format

```
<drop type template statement> ::=
      DROP TYPE TEMPLATE <specific type template designator> <drop behavior>
```

Syntax Rules

1) Let *ST* be the type template identified by the <specific type template designator> and let *TF* be the type template family of *ST*.

2) If RESTRICT is specified, then no <abstract data type definition> generated from *TT* shall be referenced in any <table definition>, <view definition>, <abstract data type definition>, <routine>, <module>, or <type template definition>.

   Note: <routine> is defined in Subclause 11.3, "<routine>", in Part 4 of this
   ANSI American
   ISO International
   Standard.

Access Rules

1) The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <type template name> immediately contained in the <specific type template designator>.

General Rules

1) Let *A* be the current <authorization identifer>. Let *STTD* be the <specific type template designator>. The following <revoke statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   REVOKE ALL PRIVILEGES ON *STTD* FROM *A* CASCADE

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop type template statement>.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.52 <drop data type statement>

Function

Destroy an abstract data type.

Format

```
<drop data type statement> ::=
      DROP DATA TYPE <data type name> <drop behavior>
```

Syntax Rules

1) Let *D* be the data type identified by the <data type name> and let *DN* be that <data type name>. *D* shall be an abstract data type or a distinct type.

2) *D* shall not be a temporary abstract data type.

3) If RESTRICT is specified, then *D* shall not be referenced in any table descriptor, view descriptor, domain descriptor, or abstract data type descriptor of an abstract data type other than *D*, or distinct type descriptor of a distinct type other than *D*.

4) Let *R* be any routine that references *D*.

5) If RESTRICT is specified, then *R* shall not be referenced in the <query expression> of any view descriptor, the <search condition> of any constraint descriptor, the <default option> of any column descriptor, or in any trigger descriptor, domain descriptor, assertion descriptor, or abstract data type descriptor of an abstract data type other than *D*, or distinct type descriptor of a distinct type other than *D*.

6) The schema identified by the explicit or implicit schema name of *DN* shall include the descriptor of *D*.

   Note: If CASCADE is specified, then such referenced objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

1) The <authorization identifier> of the current SQL-session shall be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of *D*.

General Rules

1) Let *P* be the <external function name> of any <external function declaration> that references *D*. The following <drop external function statement> is effectively executed for each such <external function declaration> without further Access Rule checking:

   DROP EXTERNAL *P* CASCADE

****Editor's Note****
The "<drop external function statement>" in the previous Rule is a reference to an undefined production. Have I missed something?

2) For every privilege descriptor that references *D*, the following <revoke statement> is effectively executed:

   REVOKE *PRIV* ON *D* FROM *GRANTEE* CASCADE

   where *PRIV* and *GRANTEE* are respectively the action and grantee in the privilege descriptor.

3) The identified data type and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop data type statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.53 <operators definition>

Function

Define operators.

Format

```
<operators definition> ::=
       CREATE <operators definition body>

<operators definition body> ::=
       OPERATORS <operator group>...

<operator group> ::=
       <operator form> <operator level> <operator list>

<operator form> ::=
         PREFIX
       | INFIX
       | POSTFIX

<operator level> ::=
       10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19

<operator list> ::=
       ( <operator> [ { <comma> <operator> }... ] )

<operator> ::=
         <operator identifier>
       | <operator symbol>

<operator identifier> ::=
       <schema qualified name>

<operator symbol> ::=
       <operator symbol character>...

<operator symbol character> ::=
         <percent>
       | <ampersand>
       | <circumflex>
       | <asterisk>
       | <plus sign>
       | <minus sign>
       | <solidus>
       | <less than operator>
       | <equals operator>
       | <greater than operator>
       | <vertical bar>

<SQL infix operator symbol> ::=
         <asterisk>
       | <plus sign>
       | <minus sign>
       | <solidus>
       | <less than operator>
       | <equals operator>
       | <greater than operator>
```

```
        | <concatenation operator>
        | <not equals operator>
        | <less than or equals operator>
        | <greater than or equals operator>

<SQL prefix operator symbol> ::=
          <plus sign>
        | <minus sign>
```

Syntax Rules

1) For an <operator definition> *OD*, an *infix user operator symbol*, *prefix user operator symbol*, or *postfix user operator symbol* is an <operator symbol> that is contained in an <operator group> of *OD* whose <operator form> is INFIX, PREFIX, or POSTFIX, respectively. An *infix operator symbol* is either an infix user operator symbol or an <SQL infix operator symbol>. A *prefix operator symbol* is either a prefix user operator symbol or an <SQL prefix operator symbol>. A *postfix operator symbol* is a postfix user operator symbol.

2) An *operator symbol* is a string of <operator symbol character>s. For an operator string *OSS*, a *postfix-infix-prefix partition* of *OD* is a partitioning of *OSS* into substrings C_1, C_2, . . . ,C_n such that for some j in the range 1 through n inclusive, C_j is an infix operator symbol and for all i in the range 1 through $j-1$, inclusive, C_i is a postfix operator. amd for all k in the range from $j+1$ through n, inclusive, C_k is a prefix operator symbol.

   A *postfix-infix partition* of *OSS* is a partitioning of *OD* into substrings C_1, C_2, . . . ,C_n such that C_n is an infix operator symbol and for all i in the range from 1 to $n-1$), inclusive, C_i is a postfix operator symbol.

   An *infix-prefix partition* of *OSS* is a partitioning of *OD* into substrings C_1, C_2, . . . ,C_n such that C_1 is an infix operator symbol and for all i in the range from 2 to n, inclusive, C_i is a prefix operator symbol.

   A *postfix partition* of *OSS* is a partitioning of *OD* into substrings C_1, C_2, . . . ,C_n such that all C_i are postfix operator symbols.

   A *prefix partition* of *OSS* is a partitioning of *OD* into substrings C_1, C_2, . . . ,C_n such that all C_i are prefix operator symbols.

   An *infix partition* of *OSS* is a partitioning of *OD* into substrings that is either a postfix-infix-prefix partition, a postfix-infix partition, or an infix-prefix partition.

3) For an <operators definition>, there shall exist no operator symbol for which there is more than one infix partition, or more than one prefix partition, or more than one postfix partition.

4) No <operator symbol> shall be the same as an <SQL infix operator symbol> or an <SQL prefix operator symbol>.

5) There shall be no <separator>s between the <operator symbol character>s of <operator symbol>.

6) No <operator symbol> shall contain two adjacent <minus sign>s. A <minus sign> shall not be the first <operator symbol character> of a prefix user operator symbol. A <minus sign> shall not be the last <operator symbol character> of a postfix user operator symbol.

7) Let *S* be an <operator symbol>. Let *DIS* be a <delimited identifier> that consists of *S* surrounded by <double quote>s. *S* is equivalent to a <schema qualified name> that is a <qualified identifier> consisting of *DIS*.

8) No <operators definition> shall contain two <operator group>s that specify both the same <operator form> and the same <operator level>.

9) An <operator> shall be contained at most once in a given <operator group>.

10) An <operator group> that specifies an <operator form> of PREFIX shall specify an <operator level> of 10 or 19.

11) An <operator group> that specifies an <operator form> of POSTFIX shall specify an <operator level> of 10 or 19.

12) An <operator identifier> shall be contained in at most one <operator group> that specifies INFIX.

13) An <operator identifier> shall be contained in at most one <operator group> that specifies PREFIX.

14) An <operator identifier> shall be contained in at most one <operator group> that specifies POSTFIX.

15) An <operator> shall be contained in at most two <operator group>s.

16) For any <schema qualified name> *SQN* in an <operators definition> in a <schema definition> *S*, let *QID* be the <qualified identifier> of *SQN*.

    a) All occurrences of *QID* in *S* shall either be contained in an <operator> or be contained in a <routine name>.

    Note: <routine name> is defined in Subclause 11.3, "<routine>", in Part 4 of this
    ANSI American
    ISO International
    Standard.

    b) For any SQL-session *SQS* whose default catalog name and default unqualified schema name identify *S*, all occurrences of *QID* in a <preparable statement> that is prepared in *SQS* by an <execute immediate statement> or a <prepare statement>, or contained in a <direct SQL statement> that is invoked directly in *SQS*, shall either be contained in an <operator> or be contained in a <routine name>.

    Note: <routine name> is defined in Subclause 11.3, "<routine>", in Part 4 of this
    ANSI American
    ISO International
    Standard.

    c) For any <module> *M* whose specified or implicit <schema name> identifies *S*, all occurrences of *QID* in *M* shall either be contained in an <operator> or be contained in a <routine name>.

    Note: <routine name> is defined in Subclause 11.3, "<routine>", in Part 4 of this
    ANSI American
    ISO International
    Standard.

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain an <operators definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.54 <grant statement>

Function

Define privileges.

Format

```
<grant statement> ::=
       GRANT <privileges>
         TO <grantee> [ { <comma> <grantee> }... ]
           [ WITH GRANT OPTION ]
```

Syntax Rules

- 1 Rule deleted.

1) Let *O* be the object identified by the <object name> contained in <privileges>.

2) Let *A* be the current <authorization identifier>. For each <grantee> specified in <privileges>, a set of privilege descriptors is identified. The privilege descriptors identified are those defining, for each <action> explicitly or implicitly in <privileges>, that <action> on *O* held by *A* with grant option.

3) If one or more <grantee>s is a role, then the WITH GRANT OPTION and <privileges> containing the <action>s REFERENCES, UNDER, or USAGE shall not be specified.

4) Grants of ALL PRIVILEGES and ALL SCHEMA PRIVILEGES <privileges> to role <grantee>s do not include the <action>s REFERENCES, UNDER, or USAGE.

5) If the <grant statement> is not contained in a <schema definition>, then the schema identified by the explicit or implicit qualifier of the <object name> shall include the descriptor of *O*. If the <grant statement> is contained in a <schema definition> *S*, then the schema identified by the explicit or implicit qualifier of the <object name> shall include the descriptor of *O* or *S* shall include a <schema element> that creates the descriptor of *O*.

Access Rules

1) The applicable privileges shall include a privilege identifying *O*.

General Rules

1) The <privileges> specify one or more privileges on the schema object as follows:

   a) ALL SCHEMA PRIVILEGES specifies all privileges on all <schema element>s in the containing schema, including <schema element>s subsequently added by <SQL schema statement>.

   b) The <object privileges> specify one or more privileges on the object identified by the <object name>.

   c) EXECUTE specifies the execute privilege on the specified module or external routine.

2) For every identified privilege descriptor, a privilege descriptor is created that specifies the identical <grantee>, <action>, object *O*, and grantor *A*. Let *CPD* be the set of privilege descriptors created.

3) For every identified privilege descriptor whose action is SELECT, INSERT, UPDATE, or REFERENCES without a column name, privilege descriptors are also created for each column *C* in *O* for which *A* holds the corresponding privilege with grant option. For each such column, a privilege descriptor is created that specifies the identical <grantee>, the identical <action>, object *C*, and grantor *A*.

   If <privileges> contains an <action> that is DELETE or TRIGGER, then a table privilege descriptor is created. If <privileges> contains an <action> that is SELECT, INSERT, UPDATE, or REFERENCES without a <privilege column list>, then a table privilege descriptor is created along with a column privilege descriptor for each column in the table identified by <table name>. If <privileges> contains an <action> that is SELECT, INSERT, UPDATE, or REFERENCES with a <privilege column list>, then a column privilege descriptor is created for each column in the <privilege column list>. If <privileges> contains an <action> that is USAGE, then a usage privilege descriptor is created.

   ****Editor's Note****

   Something is odd about the two paragraphs that make up General Rule 3. The first only cares about identified privilege descriptors (those defining, for each <action explicitly or implicitly in <privileges>, that <action> on *O* held by *A* with grant option). But the second does not consider whether or not a given privilege descriptor is identified. Is one to presume that an implementation is to generate privilege descriptors for DELETE or TRIGGER even if the current user doesn't have grant privilege against those privileges? Or, based on the second sentence of the second paragraph, create SELECT or INSERT privilege descriptors that the first paragraph expressly wouldn't have created?!

4) If WITH GRANT OPTION was specified, each privilege descriptor also indicates that the privilege is grantable.

5) If <table name> is specified, then let *T* be the table identified by the <table name>.

6) For every inherently updatable view *V* owned by some grantee *G* such that *T* is some leaf underlying table of the <query expression> of *V*:

   a) Let *VN* be the <table name> of *V*.

   b) If WITH GRANT OPTION is specified, then let *WGO* be "WITH GRANT OPTION"; otherwise, let *WGO* be a zero-length string.

   c) For every privilege descriptor *PD* in *CPD*, let *PA* be the action included in *PD*.

      i) If *PA* is INSERT, UPDATE, or DELETE, then the following <grant statement> is effectively executed as though the current <authorization identifier> were "_SYSTEM" and without further Access Rule checking:

         GRANT *PA* ON *VN* TO *G WGO*

      ii) If *PA* is *A*(*CT*), where *A* is INSERT or UPDATE, and *CT* is the name of some column of *T* such that there is a corresponding column in *V*, named *CVN*, that is derived from *CT*, then the following <grant statement> is effectively executed as though the current <authorization identifier> were "_SYSTEM" and without further Access Rule checking:

         GRANT *A*(*CVN*) ON *VN* TO *G WGO*

7) For every <grantee> *G* and for every domain *D1* owned by *G*, if the user privileges of *G* contain the REFERENCES privilege WITH GRANT OPTION on every column referenced in the <search condition> included in a domain constraint descriptor included in the domain descriptor of *D1* and a grantable USAGE privilege on all domains, character sets, collations, and translations whose <domain name>s, <character set name>s, <collation name>s, and <translation name>s, respectively, are included in the domain descriptor, then for every privilege descriptor with <privileges> USAGE, a grantor of "_SYSTEM", object *D1*, and <grantee> *G* that is not grantable, the following <grant statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   GRANT USAGE ON DOMAIN *D1* TO *G* WITH GRANT OPTION

8) For every <grantee> *G* and for every collation *C1* owned by *G*, if the user privileges of *G* include a grantable USAGE privilege for the character set name included in the collation descriptor of *C1* and a grantable USAGE privilege for the translation name, if any, included in the collation descriptor of *C1*, then for every privilege descriptor with a <privileges> USAGE a grantor of "_SYSTEM", object of *C1*, and <grantee> *G* that is not grantable, the following <grant statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   GRANT USAGE ON COLLATION *C1* TO *G* WITH GRANT OPTION

9) For every <grantee> *G* and for every translation *T1* owned by *G*, if the user privileges of *G* contain a grantable USAGE privilege for every character set identified by a <character set specification> contained in the <translation definition> of *T1*, then for every privilege descriptor with a <privileges> *P*, a grantor of "_SYSTEM", object of *T1*, and <grantee> *G* that is not grantable, the following <grant statement> is effectively executed as though the current <authorization identifier> were "_SYSTEM" and without further Access Rule checking:

   GRANT *P* ON TRANSLATION *T1* TO *G* WITH GRANT OPTION

10) If <table name> is specified, then for each view *V* owned by some <grantee> *G* such that *T* or some column *CT* of *T* is referenced in the <query expression> of *V*, let RT_i, for *i* ranging from 1 to the number of tables identified by the <table reference>s contained in the <query expression> of *V*, be the <table name>s of those tables. For every column *CV* of *V*:

    a) Let CRT_{ij}, for *j* ranging from 1 to the number of columns of RT_i that are underlying columns of *CV*, be the <column name>s of those columns.

    b) If WITH GRANT OPTION was specified, then let *WGO* be "WITH GRANT OPTION"; otherwise, let *WGO* be a zero-length string.

    c) If, following successful execution of the <grant statement>, the user privileges of *G* will contain a REFERENCES(CRT_{ij}) for all *i* and for all *j*, and will contain a REFERENCES privilege on some column of RT_i for all *i*, then the following <grant statement> is effectively executed as though the current <authorization identifier> were "_SYSTEM" and without further Access Rule checking:

       GRANT REFERENCES (*CV*) ON *V* TO *G WGO*

    d) If, following successful execution of the <grant statement>, the user privileges of *G* will contain a SELECT(CRT_{ij}) for all *i* and for all *j*, and will contain a SELECT privilege on some column of RT_i for all *i*, then the following <grant statement> is effectively executed as though the current <authorization identifier> were "_SYSTEM" and without further Access Rule checking:

       GRANT SELECT (*CV*) ON *V* TO *G WGO*

11) If two privilege descriptors are identical except that one indicates that the privilege is grantable and the other indicates that the privilege is not grantable, then both privilege descriptors are set to indicate that the privilege is grantable.

12) Redundant duplicate privilege descriptors are removed from the multiset of all privilege descriptors.

13) For every combination of <grantee> and <action> on *O* specified in <privileges>, if there is no corresponding privilege descriptor in the set of identified privilege descriptors, then a completion condition is raised: *warning—privilege not granted*.

14) If ALL PRIVILEGES was specified, then for each grantee, if no privilege descriptors were identified, then a completion condition is raised: *warning—privilege not granted*.

15) The privileges defined for *T* do not affect the privileges of any supertable or any subtable of *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

ANSI Only–SQL3

a) In Conforming Full SQL language, an <object name> shall not specify SET, MULTISET, or LIST.

ISO Only—caused by ANSI changes not yet considered by ISO

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) In Conforming Intermediate SQL language, an <object name> shall not specify COLLATION or TRANSLATION.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) In Conforming Entry SQL language, an <object name> shall not specify TABLE.

b) In Conforming Entry SQL language, an <object name> shall not specify CHARACTER SET or DOMAIN.

11.55 <role definition>

Function

Define a role.

Format

```
<role definition> ::= CREATE ROLE <role name>
```

Syntax Rules

1) The specified <role name> shall be different from any other <role name> or <authorization identifier> in the SQL environment.

Access Rules

1) The Access Rules are implementation-defined.

General Rules

1) A <role definition> defines a role.

2) Let *B* be the <authorization identifier> of the current SQL-session. A role authorization descriptor is created that identifies that the role identified by <role name> has been granted to *B* with the WITH
ANSI GRANT
ISO ADMIN
OPTION.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <role definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.56 <grant role statement>

Function

Define role authorizations.

Format

```
<grant role statement> ::=
       GRANT <role granted> [ { <comma> <role granted> }... ]
         TO <grantee> [ { <comma> <grantee> }... ]
```

ANSI Only--SQL3

```
    [ WITH GRANT OPTION ]
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
    [ WITH ADMIN OPTION ]
```

```
<role granted> ::= <role name>
```

Syntax Rules

1) No role identified by a specified <grantee> shall be contained in any role identified by a specified <role granted>; that is, no cycles of role grants are allowed.

Access Rules

1) The applicable <authorization identifier> shall have all roles identified by the specified <role granted>s as applicable roles with the WITH
ANSI GRANT
ISO ADMIN
OPTION.

General Rules

1) For every <grantee> specified, a set of role authorization descriptors is created that defines the grant of each role identified by a <role granted> to the <grantee>.

2) If WITH
ANSI GRANT
ISO ADMIN
OPTION is specified, then each role authorization descriptor also indicates that the role is grantable with the WITH
ANSI GRANT
ISO ADMIN
OPTION.

3) If two role authorization descriptors are identical except that one indicates that the role is grantable with the WITH
ANSI GRANT
ISO ADMIN
OPTION and the other indicates that the role is not, then both role authorization descriptors are set to indicate that the role is grantable with the WITH
ANSI GRANT
ISO ADMIN
OPTION.

4) Redundant duplicate role authorization descriptors are removed from the multiset of all role authorization descriptors.

****Editor's Note****

The above Rules make it permissible to grant a role to another role WITH
ANSI GRANT
ISO ADMIN
OPTION. This in turn allows an <authorization identifier> to grant a role even if the role is not specifically granted to the <authorization identifier>. This has been identified as a possible problem (see Possible Problem **310** in the Editors Notes).

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <grant role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.57 <revoke role statement>

Function

Destroy role authorizations.

Format

```
<revoke role statement> ::=
      REVOKE <role revoked> [ { <comma> <role revoked> }... ]
        FROM <grantee> [ { <comma> <grantee> }... ]

<role revoked> ::= <role name>
```

Syntax Rules

1) For every <grantee> specified, the set of role authorization descriptors is identified that defines the grants of the specified <role revoked>s to the <grantee>.

Access Rules

1) At least one of the current <authorization identifier> and the enabled applicable roles shall have a role authorization identifier which authorizes a role with the WITH
ANSI GRANT
ISO ADMIN
OPTION for every role identified by a <role revoked>.

2) For every role identified by a <role revoked> there shall exist an <authorization identifier> where the role authorization identifiers which define that <authorization identifier>s applicable roles and which are not identified by the <revoke role statement> include at least one with the WITH
ANSI GRANT
ISO ADMIN
OPTION on the role identified.

General Rules

1) The identified role authorization descriptors are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <revoke role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.58 <drop role statement>

Function

Destroy a role.

Format

```
<drop role statement> ::= DROP ROLE <role name>
```

Syntax Rules

None.

Access Rules

1) At least one of the current <authorization identifier> and the enabled applicable roles shall have a role authorization identifier which authorizes a role with the WITH
ANSI GRANT
ISO ADMIN
OPTION for every role identified by a <role revoked>.

General Rules

1) Let *R* be the role identified by the specified <role name>.

2) Let *A* be any <authorization identifier> or role identified by a role authorization descriptor as having been granted *R*.

3) The following <revoke role statement> is effectively executed:

   REVOKE *R* FROM *A*

4) The identified role and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <drop role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.59 <revoke statement>

Function

Destroy privileges.

Format

```
<revoke statement> ::=
      REVOKE [ GRANT OPTION FOR ] <privileges>
        FROM <grantee> [ { <comma> <grantee> }... ] <drop behavior>
```

Syntax Rules

1) SELECT is equivalent to specifying both the SELECT table privilege and SELECT (<privilege column list>) for all columns of <table name>.

2) INSERT is equivalent to specifying both the INSERT table privilege and INSERT (<privilege column list>) for all columns of <table name>.

3) UPDATE is equivalent to specifying both the UPDATE table privilege and UPDATE (<privilege column list>) for all columns of <table name>.

4) REFERENCES is equivalent to specifying both the REFERENCES table privilege and REFERENCES (<privilege column list>) for all columns of <table name>.

5) Let *O* be the object identified by the <object name>.

6) Let *A* be the current <authorization identifier>. For every <grantee> specified, a set of privilege descriptors is identified. A privilege descriptor *P* is said to be *identified* if it belongs to the set of privilege descriptors that define, for any <action> explicitly or implicitly in <privileges>, that <action> on *O* granted by *A* to <grantee>.

   Note: Column privilege descriptors become identified when <action> explicitly or implicitly contains a <privilege column list>.

7) A privilege descriptor *D* is said to be *directly dependent on* another privilege descriptor *P* if either:

   a) The following conditions hold:

      i) *P* indicates that the privilege that it represents is grantable, and

      ii) The grantee of *P* is the same as the grantor of *D* or the grantee of *P* is PUBLIC, and

      iii) Case:

         1) *P* and *D* are both column privilege descriptors. The action and the identified column of *P* are the same as the action and identified column of *D*, respectively.

         2) Neither *P* nor *D* are column privilege descriptors. The action and the identified table, domain, character set, collation, or translation of *P* are the same as the action and the identified table, domain, character set, collation, or translation of *D*, respectively.

b) The following conditions hold:

i) The privilege descriptor for *D* indicates that its grantor is the special grantor value "_SYSTEM", and

ii) The action of *P* is the same as the action of *D*, and

iii) The grantee of *P* is the owner of the table, collation, or translation identified by *D*, or the grantee of *P* is PUBLIC, and

iv) One of the following conditions hold:

1) *P* and *D* are both table privilege descriptors, the privilege descriptor for *D* identifies the <table name> of an inherently updatable view *V* and the identified table of *P* is the underlying table of the <query expression> of *V*.

2) *P* and *D* are both column privilege descriptors, the privilege descriptor *D* identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V* and either:

A) *V* is an inherently updatable view. For every column *CV* identified by a <column name> *CVN*, there is a corresponding column in the underlying table of the <query expression> *TN*. Let *CTN* be the <column name> of the column of the <query expression> from which *CV* is derived. The action for *P* is UPDATE or INSERT and the identified column of *P* is *TN.CTN*; or

B) For every table *T* identified by a <table reference> contained in the <query expression> of *V* and for every column *CT* that is a column of *T* and an underlying column of *CV*, the action for *P* is REFERENCES and either the identified column of *P* is *CT* or the identified table of *P* is *T*; or

C) For every table *T* identified by a <table reference> contained in the <query expression> of *V* and for every column *CT* that is a column of *T* and an underlying column of *CV*, the action for *P* is SELECT and either the identified column of *P* is *CT* or the identified table of *P* is *T*.

3) The privilege descriptor *D* identifies the <collation name> of a <collation definition> *CO* and the identified character set name of *P* is included in the collation descriptor for *CO* or the identified translation name of *P* is included in the collation descriptor for *CO*.

4) The privilege descriptor *D* identifies the <translation name> of a <translation definition> *TD* and the identified character set name of *P* is contained in the <source character set specification> or the <target character set specification> immediately contained in *TD*.

c) The following conditions hold:

i) The privilege descriptor for *D* indicates that its grantor is the special grantor value "_SYSTEM";

ii) The grantee of *P* is the owner of the domain identified by *D*, or the grantee of *P* is PUBLIC, and

iii) The privilege descriptor *D* identifies the <domain name> of a <domain definition> *DO* and either the column privilege descriptor *P* has an action of REFERENCES and identifies a column referenced in the <search condition> included in the domain descriptor for *DO*, or the privilege descriptor *P* has an action of USAGE and identifies a domain, collation, character set, or translation whose <domain name>, <collation name>, <character set name> or <translation name>, respectively, is contained in the <search condition> of the domain descriptor for *DO*.

8) The *privilege dependency graph* is a directed graph such that:

a) Each node represents a privilege descriptor, and

b) Each arc from node *P1* to node *P2* represents the fact that *P2* directly depends on *P1*.

An *independent node* is one that has no incoming arcs.

9) A privilege descriptor *P* is said to be *modified* if:

a) *P* indicates that the privilege that it represents is grantable, and

b) *P* directly depends on an identified privilege descriptor or a modified privilege descriptor, and

c) Case:

i) The following conditions hold:

1) *P* is neither a SELECT nor a REFERENCES column privilege descriptor that identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*, and;

2) Let *XO* and *XA* respectively be the identifier of the object identified by a privilege descriptor *X* and the action of *X*. Within the set of privilege descriptors upon which *P* directly depends, there exist some *XO* and *XA* for which the set of identified privilege descriptors unioned with the set of modified privilege descriptors include all privilege descriptors specifying the grant of *XA* on *XO* WITH GRANT OPTION.

ii) The following conditions hold:

1) *P* is a column privilege descriptor that identifies a column *CV* named by a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V* with an action *PA* of REFERENCES or SELECT; and

2) Let *SP* be the set of privileges upon which *P* directly depends. For every table *T* identified by a <table reference> contained in the <query expression> of *V*, let *RT* be the <table name> of *T*. There exists a column *CT* whose <column name> is *CRT*, such that:

A) *CT* is a column of *T* and an underlying column of *CV*, and

B) Every privilege descriptor *PD* that is the descriptor of some member of *SP* that specifies the action *PA* on *CRT* WITH GRANT OPTION is either an identified privilege descriptor for *CRT* or a modified privilege descriptor for *CRT*.

and

d) At least one of the following is true:

i) GRANT OPTION FOR is specified and the grantor of *P* is the special grantor value "_SYSTEM".

ii) There exists a path to *P* from an independent node that includes no identified or modified privilege descriptors. *P* is said to be a *marked modified privilege descriptor*.

iii) *P* directly depends on a marked modified privilege descriptor, and the grantor of *P* is the special grantor value "_SYSTEM". *P* is said to be a *marked modified privilege descriptor*.

10) A privilege descriptor *P* is *abandoned* if:

Case:

a) It is not an independent node, and *P* is not itself an identified or a modified privilege descriptor, and there exists no path to *P* from any independent node other than paths that include an identified privilege descriptor or a modified privilege descriptor.

b) The following conditions hold:

i) *P* is a column privilege descriptor that identifies a <column name> *CVN* explicitly or implicitly contained in the <view column list> of a <view definition> *V*, with an action *PA* of REFERENCES or SELECT; and

ii) Letting *SP* be the set of privileges upon which *P* directly depends, either:

1) There exists some table name *RT* such that:

A) *RT* is the name of the table identified by some <table reference> contained in the <query expression> of *V*, and

B) For every column privilege descriptor *CPD* that is the descriptor of some member of *SP* that specifies the action *PA* on *RT*, *CPD* is either an identified privilege descriptor for *RT* or an abandoned privilege descriptor for *RT*.

or

2) There exists some column name *CRT* such that:

A) *CRT* is the name of some column of the table of some <table reference> contained in the <query expression> of *V*, and

B) For every column privilege descriptor *CPD* that is the descriptor of some member of *SP* that specifies the action *PA* on *CRT*, *CPD* is either an identified privilege descriptor for *CRT* or an abandoned privilege descriptor for *CRT*.

11) Let *S1* be the name of any schema and let *A1* be the applicable <authorization identifier> for privilege determination with the schema identified by *S1*.

12) Let *V* be any view descriptor included in *S1*. *V* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having its user privileges SELECT privilege on one or more tables or USAGE privilege on one or more domains, abstract data types, collations, character sets, or translations or EXTERNAL PRIVILEGES on one or more of the external function invocations whose names are contained in the <query expression> of *V*.

13) Let *TC* be any table constraint descriptor included in *S1*. *TC* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having its user privileges REFERENCES privilege on one or more referenced columns of *TC* or USAGE privilege on one or more domains, abstract data types, collations, character sets, or translations or EXTERNAL PRIVILEGES on one or more of the external function invocations whose names are contained in any <search condition> of *TC*.

14) Let *AX* be any assertion descriptor included in *S1*. *AX* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having its user privileges REFERENCES privilege on one or more referenced columns of *AX* or USAGE privilege on one or more domains, abstract data types, collations, character sets, or translations or EXTERNAL PRIVILEGES on one or more of the external function invocations whose names are contained in any <search condition> of *AX*.

15) Let *TR* be any trigger descriptor included in *S1*. *TR* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having its user privileges REFERENCES privilege on one or more referenced columns of *AX* or USAGE privilege on one or more of the domains, abstract data types, collations, character sets, or translations if any, whose names are contained in any <search condition> of *TR*, the appropriate privileges for any <triggered SQL statement> appearing in *TR*, SELECT privilege on the subject table of *TR*, SELECT privilege on one or more tables appearing in any <search condition> of *TR*, or EXTERNAL PRIVILEGES on one or more of the external function invocations whose names are contained in any <search condition> of *TR*.

16) Let *DC* be any domain constraint descriptor included in *S1*. *DC* is said to be *abandoned* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having its user privileges REFERENCES privilege on one or more referenced columns of *DC* or USAGE privilege on one or more domains, abstract data types, collations, character sets, or translations whose names are contained in any <search condition> of *DC*.

17) For every domain descriptor *DO* contained in *S1*, *DO* is said to be *lost* if the destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having its user privileges EXTERNAL PRIVILEGES on one or more of the external function invocations appearing in any <search condition> of *DO*, or USAGE privilege on any character set included in the data type descriptor included in *DO*.

18) For every table descriptor *TD* contained in *S1*, for every column descriptor *CD* included in *TD*, *CD* is said to be *lost* if either:

    a) The destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors would result in *A1* no longer having in its user privileges USAGE privilege on any character set included in the data type descriptor included in *CD*, or

    b) The name of the domain *DN* included in *CD*, if any, identifies a lost domain descriptor and the destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having in its user privileges USAGE privilege on any character set included in the data type descriptor of the domain descriptor of *DN*.

19) For every module *MO*, let *G* be the <module authorization identifier> that owns *MO*. *MO* is said to be *lost* if the destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *G* no longer having its user privileges USAGE privilege on the character set referenced in the <module character set specification> of *MO*.

20) Let *DT* be an abstract data type whose descriptor is in *S1*. *DT* is said to be *abandoned* if the destruction of all identified privilege descriptors and abandoned privilege descriptors would result in <authorization identifier> *A1* no longer having in its user privileges USAGE or UNDER privilege on any abstract data type.

21) For every <type template definition> *TT* contained in *S1*, *TT* is said to be abandoned if the destruction of all identified privilege descriptors and all abandoned privilege descriptors, would result in *A1* no longer having some user privilege required to create the <type template definition>.

22) Let *SD* be the descriptor of the schema *S1*. *SD* is said to be *lost* if the destruction of all abandoned privileged descriptors, and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having in its user privileges USAGE privilege on the default character set included in the schema descriptor *SD*.

23) For every domain descriptor *DO* contained in *S1*, *DO* is said to be *impacted* if *DO* is not lost, and the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having in its user privileges USAGE privilege on the collation whose name is contained in the <collate clause> of *DO*.

24) For every column descriptor *CD* contained in a table descriptor contained in *S1*, *CD* is said to be *impacted* if *CD* is not lost, and the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having in its user privileges USAGE privilege on the collation whose name is contained in the <collate clause> of *CD*.

25) For every collation descriptor *CN* contained in *S1*, *CN* is said to be *impacted* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having in its user privileges USAGE privilege on the collation whose name is contained in the <collation source> of *CN*.

26) For every character set descriptor *CSD* contained in *S1*, *CSD* is said to be *impacted* if the destruction of all abandoned privilege descriptors and, if GRANT OPTION FOR is not specified, all identified privilege descriptors, would result in *A1* no longer having in its user privileges USAGE privilege on the collation whose name is contained in *CSD*.

27) If RESTRICT is specified, then there shall be no abandoned privilege descriptors, abandoned views, abandoned table constraints, abandoned assertions, abandoned domain constraints, lost domains, lost columns, lost modules, lost schema, impacted domains, impacted columns, impacted collations, impacted character sets, or abandoned abstract data types.

28) If CASCADE is specified, then the impact on a module that is determined to be a lost module is implementation-defined.

Access Rules

1) The applicable privileges shall include a privilege identifying *O*.

General Rules

1) If GRANT OPTION FOR is not specified, then:

   a) All abandoned privilege descriptors are destroyed.

   b) The identified privilege descriptors are destroyed.

   c) The modified privilege descriptors are set to indicate that they are not grantable.

2) If GRANT OPTION FOR is specified, then

   Case:

   a) If CASCADE is specified, then all abandoned privilege descriptors are destroyed.

   b) Otherwise, if there are any privilege descriptors directly dependent on an identified privilege descriptor that are not modified privilege descriptors, then an exception condition is raised: *dependent privilege descriptors still exist*.

   The identified privilege descriptors and the modified privilege descriptors are set to indicate that they are not grantable.

3) For every abandoned view descriptor *V*, let *S1.VN* be the <table name> of *V*. The following <drop view statement> is effectively executed without further Access Rule checking:

   DROP VIEW *S1.VN* CASCADE

4) For every abandoned table constraint descriptor *TC*, let *S1.TCN* be the <constraint name> of *TC* and let *S2.T2* be the <table name> of the table that contains *TC* (*S1* and *S2* not necessarily different). The following <alter table statement> is effectively executed without further Access Rule checking:

   ALTER TABLE *S2.T2* DROP CONSTRAINT *S1.TCN* CASCADE

5) For every abandoned assertion descriptor *AX*, let *S1.AXN* be the <constraint name> of *AX*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

   DROP ASSERTION *S1.AXN*

6) For every abandoned trigger descriptor *TR*, let *S1.TRN* be the <trigger name> of *TR*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

   DROP TRIGGER *S1.TRN*

7) For every abandoned domain constraint descriptor *DC*, let *S1.DCN* be the <constraint name> of *DC* and let *S2.DN* be the <domain name> of the domain that contains *DC*. The following <alter domain statement> is effectively executed without further Access Rule checking:

   ALTER DOMAIN *S2.DN* DROP CONSTRAINT *S1.DCN*

8) For every lost column descriptor *CD*, let *S1.TN* be the <table name> of the table whose descriptor includes the descriptor *CD* and let *CN* be the <column name> of *CD*. The following <alter table statement> is effectively executed without further Access Rule checking:

   ALTER TABLE *S1.TN* DROP COLUMN *CN* CASCADE

9) For every lost domain descriptor *DO*, let *S1.DN* be the <domain name> of *DO*. The following <drop domain statement> is effectively executed without further Access Rule checking:

   DROP DOMAIN *S1.DN* CASCADE

10) For every lost schema descriptor *SD*, the default character set of that schema is modified to include the name of the implementation-defined <character set specification> that would have been this schema's default character set had the <schema definition> not specified a <schema character set specification>.

11) If the object identified by *O* is a collation, let *OCN* be the name of that collation.

12) For every impacted domain descriptor *DO*, *DO* is modified such that it does not include *OCN*.

13) For every impacted column descriptor *CD*, *CD* is modified such that it does not include *OCN*.

14) For every impacted collation descriptor *CN*, *CN* is modified such that it does not include *OCN*. If *CN* does not include any translation name, then *CN* is modified to indicate that it utilizes the DEFAULT collation for its character repertoire.

15) For every impacted character set descriptor *CSD*, *CSD* is modified such that it does not include *OCN*. If *CSD* does not include any translation name, then *CSD* is modified to indicate that it utilizes the DEFAULT collation for its character repertoire.

16) For every abandoned abstract data type *DT* with <abstract data type name> *S1.DTN*, the following <drop data type statement> is effectively executed without further Access Rule checking:

   DROP DATA TYPE *S1.DTN* CASCADE

17) For every abandoned type template *TT* with <type template name> *S1.TTN*, the following <drop type template statement> is effectively executed without further Access Rule checking:

   DROP TYPE *S1.TTN* CASCADE

18) For every combination of <grantee> and <action> on *O* specified in <privileges>, if there is no corresponding privilege descriptor in the set of identified privilege descriptors, then a completion condition is raised: *warning—privilege not revoked*.

19) If ALL PRIVILEGES was specified, then for each <grantee>, if no privilege descriptors were identified, then a completion condition is raised: *warning—privilege not revoked*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain a <revoke statement>.

ANSI Only–SQL3

11.60 <default role definition>

Function

Define a default role.

Format

```
<default role definition> ::=
       CREATE DEFAULT ROLE <role name>
```

Syntax Rules

None.

Access Rules

1) A role authorization descriptor shall exist that indicates that the role identified by <role name> has been granted to the current SQL-session <authorization identifier> or to PUBLIC.

General Rules

1) A default role descriptor will be created that identifies the role as a default role of the <authorization identifier>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <default role definition>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only–SQL3

11.61 <drop default role statement>

Function

Drop a default role.

Format

```
<drop default role statement> ::=
      DROP DEFAULT ROLE <role name>
```

Syntax Rules

None.

Access Rules

1) The <authorization identifier> shall have the role identified by the <role name> as the default role.

General Rules

1) The default role descriptor that identifies the role as a default role for the <authorization identifier> is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL: Conforming Full SQL language shall contain no <drop default role statment>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

12 Module

12.1 <SQL-client module definition>

Function

Define an SQL-client <module>.

Format

```
<SQL-client module definition> ::= <module>
```

Syntax Rules

1) Let the <module name> of an <SQL-client module definition> *MD* be the <module name>, if any, simply contained in the <module> simply contained in *MD*.

2) Case:

   a) If an <SQL-client module definition> does not contain a <module name>, then an implementation-dependent <SQL-client module name> is implicit.

   b) Otherwise, the <module name> of an <SQL-client module definition> shall be an <SQL-client module name>.

3) The <module name> of an <SQL-client module definition> shall be different from the <module name> of every other <SQL-client module definition> in the same SQL-client environment.

4) The <language clause> of the <module> simply contained in an <SQL-client module definition> shall not specify SQL.

Access Rules

None.

General Rules

1) An <SQL-client module definition> defines an SQL-client <module>.

2) The name of the SQL-client <module> is the <module name> of the <SQL-client module definition>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) An SQL-client <module> shall be associated with an SQL-agent during its execution. An SQL-agent shall be associated with at most one SQL-client module.

12.2 <module>

Function

Define a module.

Format

```
<module> ::=
       <module name clause>
         <module remainder>
       [ END MODULE ]

<module name clause> ::=
       MODULE [ <module name> ]

<module remainder> ::=
       [ <module character set specification> ]
       <language clause>
       <module authorization clause>
         [ <module path specification> ]
       <module contents>...

<module character set specification> ::=
       NAMES ARE <character set specification>

<module authorization clause> ::=
         SCHEMA <schema name>
       | AUTHORIZATION <module authorization identifier>
       | SCHEMA <schema name> AUTHORIZATION <module authorization identifier>

<module authorization identifier> ::=
       <authorization identifier>

<module path specification> ::=
       PATH <schema name list>

<module contents> ::=
         <global declaration> [ <semicolon> ]
```

- 1 alaternative moved to Part 4

```
<global declaration> ::=
         <declare cursor>
       | <temporary abstract data type declaration>
       | <temporary table declaration>
       | <temporary view declaration>
```

Syntax Rules

1) If the <language clause> of the containing <module> specifies ADA, then a <module name> shall be specified, and that <module name> shall be a valid Ada library unit name.

2) If SCHEMA <schema name> is not specified, then a <schema name> equal to <module authorization identifier> is implicit.

3) If a <module character set specification> is not specified, then a <module character set specification> that specifies an implementation-defined character set that contains at least every character that is in <SQL language character> is implicit.

4) If <module path specification> is not specified, then a <module path specification> containing an implementation-defined <schema name list> that includes the <schema name> contained in <module authorization clause> is implicit.

5) Each <schema name> contained in <schema name list> shall conform to the Format and Syntax Rules of <schema name>.

6) The explicit or implicit <catalog name> of each <schema name> contained in <schema name list> shall be the same as the <catalog name> of the <schema name> contained in <module authorization clause>.

7) No <schema name> in <schema name list> shall be specified more than once.

8) If the explicit or implicit <schema name> does not specify a <catalog name>, then an implementation-defined <catalog name> is implicit.

9) The implicit or explicit <catalog name> is the implicit <catalog name> for all unqualified <schema name>s in the <module>.

   Note: The omission of <semicolon> in <module contents> is a deprecated feature that is supported for compatibility with earlier versions of this
   ANSI American
   ISO International
   Standard. See Annex D, "Deprecated features".

10) A <declare cursor> shall precede in the text of the <module> any <routine> that references the <cursor name> of the <declare cursor>.

11) For every <declare cursor> in a <module>, the <module> shall contain at least one <open statement> that specifies the <cursor name> declared in the <declare cursor>.

    Note: See the Syntax Rules of Subclause 13.1, "<declare cursor>".

Access Rules

None.

General Rules

1) If the SQL-agent performs calls of <routine>s from more than one Ada task, then the results are implementation-dependent.

2) Case:

   a) If a <module authorization identifier> is specified, then it is the current <authorization identifier> for privilege determination for the execution of each <routine> in the <module>.

   b) Otherwise, the current <authorization identifier> for privilege determination for the execution of each <routine> in the <module> is the SQL-session <authorization identifier>.

3) The explicit or implicit <schema name list> is used as the default SQL-path of the <module>. The default SQL-path is used to effectively qualify unqualified <routine name>s that are immediately contained in <routine invocation>s that are contained in the <module>.

4) After the last time that an SQL-agent performs a call of a <routine>:

   a) A <rollback statement> or a <commit statement> is effectively executed. If an unrecoverable error has occurred, or if the SQL-agent terminated unexpectedly, or if any constraint is not satisfied, then a <rollback statement> is performed. Otherwise, the choice of which of these SQL-statements to perform is implementation-dependent. If the implementation choice is <commit statement>, then all holdable-cursors are first closed. The determination of whether an SQL-agent has terminated unexpectedly is implementation-dependent.

   b) All SQL-sessions associated with the SQL-agent are terminated.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <module contents> shall not specify a <temporary view declaration>, or a <routine> that specifies an external routine.

   b) Conforming Full SQL language shall not contain any <module path specification>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <module> shall not contain a <temporary table declaration>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <module character set specification> shall not be specified.

   b) A <module authorization clause> shall specify AUTHORIZATION and shall not specify SCHEMA.

• 1 Subclause moved to Part 4

12.3 <SQL procedure statement>

Function

Define all of the SQL-statements that are <SQL procedure statement>s.

Format

```
<SQL procedure statement> ::=
       [ ASYNC <left paren> <async statement identifier> <right paren> ]
             <SQL executable statement>

<async statement identifier> ::= <numeric value expression>

<SQL executable statement> ::=
         <SQL schema statement>
       | <SQL data statement>
       | <SQL transaction statement>
       | <SQL connection statement>
       | <SQL session statement>
       | <SQL diagnostics statement>

<SQL schema statement> ::=
         <SQL schema definition statement>
       | <SQL schema manipulation statement>

<SQL schema definition statement> ::=
         <schema definition>
       | <table definition>
       | <view definition>
       | <grant statement>
       | <role definition>
       | <grant role statement>
       | <domain definition>
       | <null class definition>
       | <character set definition>
       | <collation definition>
       | <translation definition>
       | <assertion definition>
       | <trigger definition>

• 1 alternative moved to Part 4
       | <abstract data type definition>
       | <distinct type definition>
       | <type template definition>

<SQL schema manipulation statement> ::=
         <drop schema statement>
       | <alter table statement>
       | <drop table statement>
       | <drop view statement>
       | <revoke statement>
       | <revoke role statement>
       | <drop role statement>
       | <alter domain statement>
       | <drop domain statement>
       | <drop null class statement>
       | <drop character set statement>
       | <drop collation statement>
       | <drop translation statement>
```

```
       | <drop assertion statement>
       | <drop trigger statement>
```

- 1 alternative moved to Part 4

```
       | <drop data type statement>
       | <drop type template statement>

<SQL data statement> ::=
         <open statement>
       | <fetch statement>
       | <close statement>
       | <select statement: single row>

| <free locator statement>
```

- 1 alternative deleted.

- 1 alternative deleted.

```
       | <SQL data change statement>

<SQL data change statement> ::=
         <delete statement: positioned>
       | <delete statement: searched>
       | <insert statement>
       | <update statement: positioned>
       | <update statement: searched>

<SQL transaction statement> ::=
         <start transaction statement>
       | <set transaction statement>
       | <set constraints mode statement>
       | <test completion statement>
       | <savepoint statement>
       | <release savepoint statement>
       | <commit statement>
       | <rollback statement>

<SQL connection statement> ::=
         <connect statement>
       | <set connection statement>
       | <disconnect statement>

<SQL session statement> ::=
         <set session authorization identifier statement>
       | <set role statement>
       | <set local time zone statement>

<SQL diagnostics statement> ::=
       <get diagnostics statement>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) An *atomic execution context* is active during execution of an <SQL procedure statement>. When this statement completes, all savepoints that have been established during its execution are destroyed.

2) If any exception condition is raised during the execution of <SQL procedure statement> *S*, either by *S* or any triggered <SQL procedure statement>, then:

   a) One of the exception conditions is the active condition.

   b) All changes made to SQL-data or schemas by the execution of *S* and any <SQL procedure statement>s triggered by the execution of *S* are canceled.

   c) Diagnostics information resulting from the execution of *S* is placed into the diagnostics area as specified in Subclause 17.1, "<get diagnostics statement>".

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <SQL procedure statement> shall not specify ASYNC.

   b) An <SQL schema definition statement> shall not be a <role definition>, <grant role statement>, <null class definition>, <trigger definition>, <routine>, <abstract data type definition>, <distinct type definition>, or <type template definition>.

   c) An <SQL shema manipulation statement> shall not be a <revoke role statement>, <drop role statement>, <drop null class statement>, <drop trigger statement>, <drop routine statement>, <drop data type statement>, or <drop type template statement>.

   d) An <SQL transaction statement> shall not be a <test completion statement>, <savepoint statement>, or <release savepoint statement>.

   e) An <SQL session statement> shall not be a <set role statement>.

   f) An <SQL transaction statement> shall not be a <start transaction statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) An <SQL procedure statement> shall not be an <SQL schema definition statement>.

• 1 Subclause moved to Part 4

12.4 Rules for externally-invoked <routine>s

Function

Specify General Rules for externally-invoked <routine>s.

General Rules

1) A <routine> defines a procedure or function that may be called by an SQL-agent.

2) If a <routine> is contained in a <module> that is associated with an SQL-agent that is associated with another <module> that contains a <routine> with the same <routine name>, then the effect is implementation-defined.

3) If the SQL-agent that performs a call of a <routine> is not a program that conforms to the programming language standard specified by the implicit or explicit <caller language clause> of that <routine>, then the effect is implementation-dependent.

4) If the implicit or explicit <caller language clause> of a <routine> specifies ADA and the SQL-agent performs calls of <routine>s from more than one Ada task, then the results are implementation-dependent.

5) If the <module> that contains the <routine> has an explicit <module authorization identifier> *MAI* that is different from the SQL-session <authorization identifier> *SAI*, then:

   a) Whether or not *SAI* can invoke <routine>s in a <module> with explicit <module authorization identifier> *MAI* is implementation-defined, as are any restrictions pertaining to such invocation.

   b) If *SAI* is restricted from invoking a <routine> in a <module> with explicit <module authorization identifier> *MAI*, then an exception condition is raised: *invalid authorization specification*.

6) If the value of any input parameter provided by the SQL-agent falls outside the set of allowed values of the data type of the parameter, or if the value of any output parameter resulting from the execution of the <routine> falls outside the set of values supported by the SQL-agent for that parameter, then the effect is implementation-defined. If the implementation-defined effect is the raising of an exception condition, then an exception condition is raised: *data exception—invalid parameter value*.

7) Let *S* be the <SQL procedure statement> of the <routine>.

8) When the <routine> is called by an SQL-agent:

   Case:

   a) If ASYNC is specified and there are already an implementation-defined number of outstanding asynchronous <SQL procedure statement>s and *S* is not a <test completion statement> or a <rollback statement>, then an exception condition is raised: *asynchronous SQL statement not accepted*.

   b) If ASYNC is specified and the <async statement identifier> is equal to any <async statement identifier> of any <SQL procedure statement> initiated for asynchronous execution in the same SQL-transaction, the an exception condition is raised: *duplicate asynchronous SQL statement identifier*.

c) If *S* is an <SQL connection statement>, then:

i) The <module> that contains *S* is associated with the SQL-agent.

ii) The diagnostics area is emptied.

iii) *S* is executed.

iv) If *S* successfully initiated or resumed an SQL-session, then subsequent calls to a <routine> by the SQL-agent are associated with that SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

d) If *S* is an <SQL diagnostics statement>, then:

i) The <module> that contains *S* is associated with the SQL-agent.

ii) The values of all input parameters to the <routine> are established.

iii) *S* is executed.

e) Otherwise:

i) If no SQL-session is current for the SQL-agent, then

Case:

1) If the SQL-agent has not executed an <SQL connection statement> and there is no default SQL-session associated with the SQL-agent, then the following <connect statement> is effectively executed:

CONNECT TO DEFAULT

2) If the SQL-agent has not executed an <SQL connection statement> and there is a default SQL-session associated with the SQL-agent, then the following <set connection statement> is effectively executed:

SET CONNECTION DEFAULT

3) Otherwise, an exception condition is raised: *connection exception—connection does not exist*.

Subsequent calls to a <routine> by the SQL-agent are associated with the SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

ii) If an SQL-transaction is active for the SQL-agent, then *S* is associated with that SQL-transaction.

iii) If no SQL-transaction is active for the SQL-agent and *S* is a transaction-initiating SQL-statement, then

1) An SQL-transaction is effectively initiated and associated with this call and with subsequent calls of any <routine> by that SQL-agent until the SQL-agent terminates that SQL-transaction.

2) If *S* is not a <start transaction statement>, then

Case:

A) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session, then the access mode, constraint mode, and isolation level of the SQL-transaction are set as specified by the <set transaction statement>.

B) If a <set session characteristics statement> has been executed in the current SQL-session, then:

I) If that <set session characteristics statement> set the enduring transaction characteristics of access mode, then the access mode of the SQL-transaction is set to the specified access mode.

II) If that <set session characteristics statement> set the enduring transaction characteristics of isolation level, then the isolation level of the SQL-transaction is set to the specified isolation level.

III) The constraint mode for all constraints in the SQL-transaction is set to *immediate*.

C) Otherwise, the access mode of that SQL-transaction is *read-write*, the constraint mode for all constraints in that SQL-transaction is *immediate*, and the isolation level of that SQL-transaction is SERIALIZABLE.

3) The SQL-transaction is associated with the SQL-session.

4) The <module> that contains *S* is associated with the SQL-transaction.

iv) The <module> that contains *S* is associated with the SQL-agent.

v) If *S* contains an <SQL schema statement> and the access mode of the current SQL-transaction is *read-only*, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

vi) The diagnostics area is emptied.

vii) The values of all input parameters to the <routine> are established.

viii) *S* is executed as specified in Subclause 9.1, "<routine invocation>", in Part 4 of this
ANSI American
ISO International
Standard.

9) When a <routine> is called by an SQL-agent, let PD_i be the <parameter declaration> of the *i*-th parameter and let DT_i and PN_i be the <predefined type> and the <parameter name> specified in PD_i, respectively. Let PI_i be the *i*-th parameter in the procedure call.

Case:

a) If the <caller language clause> of the <routine> specifies ADA, then:

i) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i in a <general value specification> has the value PI_i.

ii) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

iii) If PN_i is used as an output parameter and no value has been assigned to PI_i, then an implementation-dependent value is assigned to PN_i.

b) If the <caller language clause> of the <routine> specifies C, then:

i) If DT_i specifies BIT(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as:

SUBSTRING (CAST (PI_i AS BIT VARYING(ML)) FROM 1 FOR L)

where ML is the implementation-defined maximum length of a BIT VARYING data type.

2) Let BL_i be the length in bits of PI_i. Let BL be the implementation-defined number of bits in a C character. Let OL be the smallest integer not less than the quotient of BL_i/BL. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER(OL))

to PI_i.

ii) If DT_i specifies CHARACTER(L) or CHARACTER VARYING(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of that value.

When such a reference is evaluated:

A) If DT_i specifies CHARACTER(L) and some C character preceding the least significant C character of the value PI_i contains the implementation-defined null character that terminates a C character string, then the remaining characters of the value are set to <space>s.

B) If DT_i specifies CHARACTER(L) VARYING, then the length in characters of the value is set to the number of characters of PI_i that precede the implementation-defined null character that terminates a C character string.

C) If the least significant C character of the value PI_i does not contain the implementation-defined null character that terminates a C character string, then an exception condition is raised: *data exception—unterminated C string*. Otherwise, that least significant C character does not correspond to any character in PI_i and is ignored.

2) Let CL_i be k greater than the maximum possible length in octets of PN_i, where k is the size in octets of the largest character in the character set of DT_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is an SQL CHARACTER(CL_i) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i, concatenated with a single implementation-defined null character that terminates a C character string.

Note: In the preceding Rule, the phrase "implementation-defined null character that terminates a C character string" implies one or more octets all of whose bits are zero and whose number is equal to the number of octets in the largest character of the character set of DT_i.

iii) If DT_i specifies BOOLEAN then . . .

> ****Editor's Note****
> Paper SLC-55a left the preceding rule "dangling" with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iv) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value PI_i.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

c) If the <caller language clause> of the <routine> specifies COBOL, then

Case:

i) If DT_i specifies BIT(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as:

SUBSTRING (CAST (PI_i AS BIT VARYING(ML)) FROM 1 FOR L)

where ML is the implementation-defined maximum length of a BIT VARYING data type.

2) Let BL_i be the length in bits of PI_i. Let BL be the implementation-defined number of bits in a COBOL character. Let OL be the smallest integer not less than the quotient of BL_i/BL. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER(OL))

to PI_i.

ii) If DT_i specifies CHARACTER(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of that value.

2) Let CL_i be the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is an SQL CHARACTER(CL_i) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i.

iii) If DT_i specifies BOOLEAN then . . .

> ****Editor's Note****
> Paper SLC-55a left the preceding rule "dangling" with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iv) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value PI_i.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

d) If the <caller language clause> of the <routine> specifies FORTRAN, then

Case:

i) If DT_i specifies BIT(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as:

SUBSTRING (CAST (PI_i AS BIT VARYING(ML)) FROM 1 FOR L)

where ML is the implementation-defined maximum length of a BIT VARYING data type.

2) Let BL_i be the length in bits of PI_i. Let BL be the implementation-defined number of bits in a Fortran character. Let OL be the smallest integer not less than the quotient of BL_i/BL. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER(OL))

to PI_i.

ii) If DT_i specifies CHARACTER(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of that value.

2) Let CL_i be the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is an SQL CHARACTER(CL_i) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i.

iii) If DT_i specifies BOOLEAN then . . .

****Editor's Note****

Paper SLC-55a left the preceding rule "dangling" with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iv) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value PI_i.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

e) If the <caller language clause> of the <routine> specifies MUMPS, then

Case:

i) If DT_i specifies CHARACTER VARYING(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of that value.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is an SQL CHARACTER VARYING(ML) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i. ML is the implementation-defined maximum length of variable-length character strings.

ii) Otherwise, DT_i specifies INT, DEC, or REAL, and:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as

CAST (PI_i AS DT_i)

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER VARYING(ML))

to PI_i, where ML is the implementation-defined maximum length of variable-length character strings.

f) If the <caller language clause> of the <routine> specifies PASCAL, then

Case:

i) If DT_i specifies BIT(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated as:

SUBSTRING (CAST (PI_i AS BIT VARYING(ML)) FROM 1 FOR L)

where ML is the implementation-defined maximum length of a BIT VARYING data type.

2) Let BL_i be the length in bits of PI_i. Let BL be the implementation-defined number of bits in a Pascal character. Let OL be the smallest integer not less than the quotient of BL_i/BL. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value

CAST (SV_i AS CHARACTER (OL))

to PI_i.

ii) If DT_i specifies CHARACTER(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of the value.

2) Let CL_i be the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value which is an SQL CHARACTER(CL_i) data type in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i.

iii) If DT_i specifies BOOLEAN then . . .

****Editor's Note****

Paper SLC-55a left the preceding rule "dangling" with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iv) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value PI_i.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

g) If the <caller language clause> of the <routine> specifies PLI, then

Case:

i) If DT_i specifies CHARACTER(L) or CHARACTER VARYING(L), then:

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i is implicitly treated an SQL character type value in the specified character set in which the octets of PI_i are the corresponding octets of the value.

2) Let CL_i be the maximum possible length in octets of PN_i. Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns a value that is:

A) if DT_i specified CHARACTER(L), then an SQL CHARACTER(CL_i) data type

B) otherwise, an SQL CHARACTER VARYING(CL_i) data type

in which octets of the value are the corresponding octets of SV_i, padded on the right with <space>s as necessary to reach the length CL_i.

ii) If DT_i specifies BOOLEAN then . . .

****Editor's Note****

Paper SLC-55a left the preceding rule "dangling" with the remark: Note to editor: this needs sorting out—but is SQL3 stuff

iii) Otherwise,

1) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i has the value PI_i.

2) Where PN_i is used as an output parameter, a reference to PN_i that assigns a value SV_i to PN_i implicitly assigns the value SV_i to PI_i.

10) If ASYNC is not specified, or if S is a <test completion statement>, a <commit statement>, or a <rollback statement>, or if the SQL-implementation has decided to execute S synchronously, then wait until the execution of S has completed.

11) If the execution of *S* resulted in an active exception condition, then any status parameters are set to the values specified for the condition in Clause 19, "Status codes".

12) If there is more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

13) If the execution of an <SQL data statement> occurs within the same SQL-transaction as the execution of an SQL-schema statement and this is not allowed by the SQL-implementation, then an exception condition is raised: *invalid transaction state— schema and data statement mixing not supported*.

14) If the cursor mode of the current SQL-transaction is cascade off and the <SQL procedure statement> is other than an <open statement>, a <fetch statement>, a <close statement>, a <select statement: single row>, or an <SQL data change statement>, then an exception condition is raised: *invalid SQL statement*.

15) Case:

    a) If *S* executed successfully, then

        i) If there is more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

        ii) Either a completion condition is raised: *successful completion*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*.

    b) If *S* did not execute successfully, then:

        i) All changes made to SQL-data or schemas by the execution of *S* are canceled.

        ii) An exception condition is raised. The status parameter(s) is (are) set to the value(s) specified for the condition in clause Clause 19, "Status codes". If there is more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

16) Case:

    a) If *S* is an <SQL diagnostics statement>, then the diagnostics area is not updated.

    b) If *S* was executed synchronously, then diagnostics information resulting from the execution of *S* is placed into the diagnostics area as specified in Clause 17, "Diagnostics management".

    c) Otherwise, no further action is taken at this time, but whenever a subsequent <test completion statement> has successfully tested the completion of *S*, then the diagnostics area is updated as required by Subclause 19.1, "SQLSTATE".

13 Data manipulation

13.1 <declare cursor>

Function

Define a cursor.

Format

```
<declare cursor> ::=
      DECLARE <cursor name> [ <cursor sensitivity> ] [ SCROLL ] CURSOR
        [ WITH HOLD ]
        FOR <cursor specification>

<cursor sensitivity> ::=
        SENSITIVE
      | INSENSITIVE

<cursor specification> ::=
      <query expression> [ <order by clause> ]
        [ <updatability clause> ]

<updatability clause> ::=
      FOR { READ ONLY | UPDATE [ OF <column name list> ] }

<order by clause> ::=
      ORDER BY <sort specification list>

<sort specification list> ::=
      <sort specification> [ { <comma> <sort specification> }... ]

<sort specification> ::=
      <sort key> [ <collate clause> ] [ <ordering specification> ]

<sort key> ::=
      <value expression>

<ordering specification> ::= ASC | DESC
```

Syntax Rules

1) If a <declare cursor> is contained in a <module> *M*, then:

   a) The <cursor name> shall be different from the <cursor name> of any other <declare cursor> in *M*.

   b) For each <open statement> *O* in the <module> that specifies the <cursor name> declared in the <declare cursor>, any <item reference> contained in the <cursor specification> shall be defined in a <parameter declaration> in the <routine> that contains *O*.

c) For each <item reference> in the <declare cursor> the data types of the parameters associated with each of the <open statement>s that specify the <cursor name> shall be the same.

Note: See the Syntax Rules of Subclause 12.2, "<module>".

2) When <cursor name> is referenced in an <update statement: positioned>, no <object column> in the <set clause> shall identify a column that is specified in a <sort specification> of an <order by clause>.

3) Let *T* be the table specified by the <query expression>.

4) Let *CS* be the cursor specified by the <cursor specification>.

5) If <updatability clause> is not specified, then:

a) If either INSENSITIVE, SCROLL, or ORDER BY is specified, or if *T* is not an inherently updatable table, then an <updatability clause> of READ ONLY is implicit.

b) Otherwise, an <updatability clause> of FOR UPDATE without a <column name list> is implicit.

6) If an <updatability clause> of READ ONLY is specified or implicit, then *CS* is not an *inherently updatable cursor*; otherwise, *CS* is an *inherently updatable cursor*.

7) If an <order by clause> is specified, then the cursor specified by the <cursor specification> is said to be an *ordered cursor*.

8) If WITH HOLD is specified, then the cursor specified by the <cursor specification> is said to be a *holdable cursor*.

9) The *simply underlying table* of a <cursor specification> is the table derived from the <query expression> simply contained in the <cursor specification>.

10) If an <order by clause> is specified, then let K_i be the <sort key> contained in the *i*-th <sort specification>. Each row of *ST* contains the values of the columns of *T* and the values of the <sort key>s for the corresponding row of *T*, computed as follows:

Case:

a) If K_i is an <unsigned integer>, then the value *N* of the <unsigned integer> shall be greater than 0 and less than or equal to the degree *T*. Let *C* be the column of *ST* whose ordinal position is *N*. The sort key value V_i is the value of *C* in a row of *ST*.

b) Otherwise, K_i is a <value expression>. The <value expression> shall not contain a <subquery> or a <set function specification> but shall contain a column reference.

i) Let *X* be any column reference directly contained in K_i.

ii) Let *QE* be the <query expression> immediately contained in the <cursor specification>. If *QE* is a <query primary> that is a <simple table> that is a <query specification>, then the <cursor specification> is said to be a *simple table query*. If the <cursor specification> is a simple table query, then let *TE* be the <table expression> contained in the <query specification> contained in *QE*.

iii) If *X* contains an <item qualifier>, then the <cursor specification> shall be a simple table query and *X* shall unambiguously identify a column of *TE*. The value of that column is used to compute the value of K_i.

iv) If *X* does not contain an <item qualifier>, then

Case:

1) If the <cursor specification> is not a simple table query, then *X* shall be identical to the name of exactly one column of *T*. The value of that column is used to compute the value of K_i.

2) Otherwise, the <cursor specification> is a simple table query.

A) *X* shall not be identical to the name of more than one column of *T*.

B) If *X* is identical to the name of a column of *T*, then the value of that column is used to compute the value of K_i.

C) If *X* is not identical to the name of any column of *T*, then *X* shall be identical to the name of exactly one column of *TE*. The value of that column is used to compute the value of K_i.

v) If the <cursor specification> is a simple table query and if the <query specification> contained in *QE* specifies the <set quantifier> DISTINCT or directly contains one or more <set function specifiction>s or if *T* is a grouped table, then there shall be a <derived column> equivalent to *X* in the <select list> of the <query specification>.

vi) The sort key value V_i is the value of K_i for a row of *T*.

11) If a <sort specification> contains a <collate clause>, then the data type of the column identified by the <sort specification> shall be character string. The column descriptor of the corresponding column in the result has the collating sequence specified in <collate clause> and the coercibility attribute *Explicit*.

12) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then INSENSITIVE shall not be specified, and either:

ISO Only–SQL3

a) There shall be one or more triggers associated with the simply underlying table of *CS* whose trigger event is DELETE and UPDATE (spanning all columns of that table), and whose trigger action time is INSTEAD OF; or

ANSI Only—caused by ISO changes not yet considered by ANSI

b) *T* shall be trigger-mutable; or

c) *T* shall be an inherently updatable table.

13) If an <updatability clause> of FOR UPDATE without a <column name list> is specified or implicit, then a <column name list> that includes the <column name> of every column of the simply underlying table of the simply underlying table of *T* implicit.

14) If an <updatability clause> of FOR UPDATE with a <column name list> is specified, then each <column name> in the <column name list> shall be the <column name> of a column of the simply underlying table of the simply underlying table of *T*.

15) If a <sort key> simply contains a <value expression> that simply contains a column reference that identifies a column whose data type is an abstract data type, then the <abstract data type definition> that defined that abstract data type shall contain a <less-than clause> that does not specify NONE.

16) The data type of a <value expression> simply contained in a <sort key> shall not be a <collection type>.

Access Rules

None.

General Rules

ISO Only–SQL3

1) If *T* is an inherently updatable table, then let *TU* be the leaf generally underlying table of the <cursor specification>.

2) If *T* is an inherently updatable table, then the cursor is associated with the named table *TU*. For every row in *T*, there is exactly one corresponding row in *TU* from which the row of *T* is derived.

ANSI Only—caused by ISO changes not yet considered by ANSI

3) If an <order by clause> is not specified, then:

   Case:

   a) If *T* is an ordered table, then the ordering of *T* is used by the cursor.

   b) Otherwise, the ordering of rows in *T* is implementation-dependent.

ISO Only–SQL3

4) If an <order by clause> is not specified, then the table specified by the <cursor specification> is *T* and the ordering of rows in *T* is implementation-dependent.

5) If an <order by clause> is specified, then the ordering of rows of the result is effectively determined by the <order by clause> as follows:

   a) Each <sort specification> specifies the sort direction for the corresponding sort key K_i. If DESC is not specified in the *i*-th <sort specification>, then the sort direction for K_i is ascending and the applicable <comp op> is the <less than operator>. Otherwise, the sort direction for K_i is descending and the applicable <comp op> is the <greater than operator>.

   b) Let *P* be any row of the result table and let *Q* be any other row of that table, and let PV_i and QV_i be the values of K_i in these rows, respectively. The relative position of rows *P* and *Q* in the result is determined by comparing PV_i and QV_i according to the rules of Subclause 8.2, "<comparison predicate>", where the <comp op> is the applicable <comp op> for K_i, with the following special treatment of null values. Whether a sort key value that is null is considered greater or less than a non-null value is implementation-defined, but all sort key values that are null shall either be considered greater than all non-null values or be considered less than all non-null values. PV_i is said to *precede* QV_i if the value of the <comparison predicate> "PV_i <comp op> QV_i" is true for the applicable <comp op>.

   c) In the result table, the relative position of row *P* is before row *Q* if and only if PV_n precedes QV_n for some *n* greater than 0 and less than the number of <sort specification>s and $PV_i = QV_i$ for all $i < n$. The relative order of two rows that are not distinct is implementation-dependent.

6) If WITH HOLD is specified and the cursor is in an open state when an SQL-transaction is terminated with a <commit statement>, then the cursor is not closed and remains open into the next SQL-transaction.

   Note: A holdable-cursor that has been held open retains its position when the new SQL-transaction is initiated. However, even if the cursor is currently positioned on a row when the SQL-transaction is terminated, before either an <update statement: positioned> or a <delete statement: positioned> is permitted to reference that cursor in the new SQL-transaction, a <fetch statement> must be issued against the cursor.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) A <declare cursor> shall not specify SENSITIVE.

   b) A <declare cursor> shall not specify WITH HOLD.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) A <declare cursor> shall not specify INSENSITIVE.

   b) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then neither SCROLL nor ORDER BY shall be specified.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <declare cursor> shall not specify SCROLL.

   b) A <cursor specification> shall not contain an <updatability clause>.

13.2 <open statement>

Function

Open a cursor and set the cursor mode of the current SQL-transaction.

Format

```
<open statement> ::=
      OPEN <cursor name>
        [ <open cascade option> ]

<open cascade option> ::=
        CASCADE ON | CASCADE OFF
```

Syntax Rules

1) Let *CR* be the cursor identified by the <cursor name> immediately contained in the <open statement>.

2) If CASCADE is not specified, then CASCADE ON is implicit.

ISO Only–SQL3

3) If CASCADE OFF is specified, then either:

   a) There shall be one or more triggers associated with the simply underlying table of *CR* whose trigger event is DELETE and INSERT and UPDATE (spanning all columns of that table), and whose trigger action time is INSTEAD OF; or

   b) *T* shall be an inherently updatable table.

ANSI Only—caused by ISO changes not yet considered by ANSI

4) Let *QE* be the <query expression> simply contained in the <declare cursor> *DC* identified by <cursor name>. Let *CR* be the cursor specified by *DC*. If CASCADE OFF is specified, then either:

   a) *QE* shall be trigger-mutable; or

   b) *CR* shall be an inherently updatable cursor.

Access Rules

1) The Access Rules for the <query expression> simply contained in the <declare cursor> identified by the <cursor name> are applied.

General Rules

1) If *CR* is not in the closed state, then an exception condition is raised: *invalid cursor state*.

2) If CASCADE OFF is specified and *CR* is a holdable-cursor, then an exception condition is raised: *invalid cursor state— cascade off incompatible with holdable-cursor*.

3) If the cursor mode of the current SQL-transaction is set to *cascade off*, then an exception condition is raised: *invalid cursor mode state*.

4) Let *S* be the <cursor specification> of cursor *CR*.

5) Cursor *CR* is opened in the following steps:

   a) A copy of *S* is effectively created in which:

      i) Each <target specification> is replaced by the value of the target;

      ii) Each <value specification> generally contained in *S* that is CURRENT_USER, SESSION_USER, or SYSTEM_USER is replaced by the value resulting from evaluation of CURRENT_USER, SESSION_USER, or SYSTEM_USER, respectively, with all such evaluations effectively done at the same instant in time; and

      iii) Each <datetime value function> generally contained in *S* is replaced by the value resulting from evaluation of that <datetime value function>, with all such evaluations effectively done at the same instant in time.

   b) Let *T* be the table specified by the copy of *S*.

   c) A table descriptor for *T* is effectively created.

   d) The General Rules of Subclause 13.1, "<declare cursor>", are applied.

   e) Case:

      i) If *S* specifies INSENSITIVE, then a copy of *T* is effectively created and cursor *CR* is placed in the open state and its position is before the first row of the copy of *T*.

      ii) Otherwise, cursor *CR* is placed in the open state and its position is before the first row of *T*.

6) Let *CC* be the set of constraints that are defined by <assertion definition>s or by <table constraint definition>s of tables that are nodes in the constraint-referential-action graph for *CR*. Let *CCL* be a comma-separated list of the <constraint name>s of the constraints in *CC*.

7) Case:

   a) If CASCADE ON is specified or implied, then:

      i) If *CR* is a cursor that was explicitly or implicitly declared FOR UPDATE and any cursor *CR1* in the current SQL-transaction that is in the open state was opened CASCADE OFF and the constraint-referential-action graphs for *CR* and *CR1* are not disjoint, then an exception condition is raised: *invalid cursor state*.

      ii) The following <set constraints mode statement> is implicitly executed:

      SET *CCL* CONSTRAINTS IMMEDIATE

iii) The cursor mode of the current SQL-transaction is set to *cascade on*.

b) If CASCADE OFF is specified, then:

i) If any cursor *CR1* that is implicitly or explicitly declared FOR UPDATE in the current SQL-transaction is in the open state and the constraint-referential-action graphs for *CR* and *CR1* are not disjoint, then an exception condition is raised: *invalid cursor state*.

ii) The following <set constraints mode statement> is implicitly executed:

SET *CCL* CONSTRAINTS DEFERRED

iii) The cursor mode of the current SQL-transaction is set to *cascade off*.

****Editor's Note****
The preceding rules deal with CASCADE ON and OFF and depend on the constraint-referential-action graph, which has been deleted. Therefore, these rules will need to be revised.

8) If *CR* specifies INSENSITIVE, and the implementation is unable to guarantee that significant changes will be invisible through *CR*, then an exception condition is raised: *cursor sensitivity exception—request rejected*.

9) If *CR* specifies SENSITIVE, and the implementation is unable to guarantee that significant changes will be visible through *CR*, then an exception condition is raised: *cursor sensitivity exception—request rejected*.

10) Whether an implementation is able to disallow significant changes that would not be visible through a currently open cursor is implementation-defined.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) An <open statement> shall not contain an <open cascade option>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

13.3 <fetch statement>

Function

Position a cursor on a specified row of a table and retrieve values from that row.

Format

```
<fetch statement> ::=
       FETCH [ [ <fetch orientation> ] FROM ]
         <cursor name> INTO <fetch target list>

<fetch orientation> ::=
         NEXT
       | PRIOR
       | FIRST
       | LAST
       | { ABSOLUTE | RELATIVE } <simple value specification>

<fetch target list> ::=
       <target specification> [ { <comma> <target specification> }... ]
```

Syntax Rules

1) If the <fetch orientation> is omitted, then NEXT is implicit.

2) Let *DC* be the <declare cursor> denoted by the <cursor name> and let *T* be the table defined by the <cursor specification> of *DC*. Let *CR* be the cursor specified by *DC*.

3) If the implicit or explicit <fetch orientation> is not NEXT, then *DC* shall specify SCROLL.

4) If a <fetch orientation> that contains a <simple value specification> is specified, then the data type of that <simple value specification> shall be exact numeric with a scale of 0.

5) Case:

   a) If the <fetch target list> contains a single <target specification> *TS* and the degree of table *T* is greater than 1, then:

      i) The data type of *TS* shall be a row type.

      ii) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to *TS* and the row type of table *T* as *TARGET* and *VALUE*, respectively.

   b) Otherwise:

      i) The number of <target specification>s in the <fetch target list> shall be the same as the degree of table *T*. The *i*-th <target specification> in the <fetch target list> corresponds with the *i*-th column of table *T*.

      ii) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to each corresponding <target specification> and column of table *T*, as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) If cursor *CR* is not in the open state, then an exception condition is raised: *invalid cursor state*.

2) If the cursor mode of the current SQL-transaction is *cascade off*, and if the previous <SQL procedure statement> that specified the <cursor name> of *CR* that was executed within the current set-processing mode session of the current SQL-transaction was a <fetch statement>, then an exception condition is raised: *invalid SQL statement*.

3) Case:

   a) If the <fetch orientation> contains a <simple value specification>, then let *J* be the value of that <simple value specification>.

   b) If the <fetch orientation> specifies NEXT or FIRST, then let *J* be +1.

   c) If the <fetch orientation> specifies PRIOR or LAST, then let *J* be −1.

4) Let T_t be a table of the same degree as *T*.

   Case:

   a) If the <fetch orientation> specifies ABSOLUTE, FIRST, or LAST, then let T_t contain all rows of *T*, preserving their order in *T*.

   b) If the <fetch orientation> specifies NEXT or specifies RELATIVE with a positive value of *J*, then:

      i) If the table *T* identified by cursor *CR* is empty or if the position of *CR* is on or after the last row of *T*, then let T_t be a table of no rows.

      ii) If the position of *CR* is on a row *R* that is other than the last row of *T*, then let T_t contain all rows of *T* ordered after row *R*, preserving their order in *T*.

      iii) If the position of *CR* is before a row *R*, then let T_t contain row *R* and all rows of *T* ordered after row *R*, preserving their order in *T*.

   c) If the <fetch orientation> specifies PRIOR or specifies RELATIVE with a negative value of *J*, then:

      i) If the table *T* identified by cursor *CR* is empty or if the position of *CR* is on or before the first row of *T*, then let T_t be a table of no rows.

      ii) If the position of *CR* is on a row *R* that is other than the first row of *T*, then let T_t contain all rows of *T* ordered before row *R*, preserving their order in *T*.

      iii) If the position of *CR* is before the next row of a row *R* that is not the last row of *T*, then let T_t contain row *R* and all rows of *T* ordered before row *R*, preserving their order in *T*.

      iv) If the position of *CR* is after the last row of *T*, then let T_t contain all rows of *T*, preserving their order in *T*.

d) If RELATIVE is specified with a zero value of J, then:

i) If the position of CR is on a row of T, then let T_t be a table comprising that one row.

ii) Otherwise, let T_t be an empty table.

5) Let N be the number of rows in T_t. If J is positive, then let K be J. If J is negative, then let K be $N+J+1$. If J is zero and ABSOLUTE is specified, then let K be zero; if J is zero and RELATIVE is specified, then let K be 1.

6) Case:

a) If K is greater than 0 and not greater than N, then CR is positioned on the K-th row of T_t and the corresponding row of T. That row becomes the current row of CR.

b) Otherwise, no SQL-data values are assigned to any targets in the <fetch target list>, and a completion condition is raised: *no data*.

Case:

i) If the <fetch orientation> specifies RELATIVE with J equal to zero, then the position of CR is unchanged.

ii) If the <fetch orientation> implicitly or explicitly specifies NEXT, specifies ABSOLUTE or RELATIVE with K greater than N, or specifies LAST, then CR is positioned after the last row.

iii) Otherwise, the <fetch orientation> specifies PRIOR, FIRST, or ABSOLUTE or RELATIVE with K not greater than N and CR is positioned before the first row.

7) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

8) If an exception condition is raised during derivation of any <derived column> associated with the current row of CR, then CR remains positioned on the current row.

9) Case:

a) If the <fetch target list> contains a single <target specification> TS and the degree of table T is greater that 1, then the current row is assigned to TS and the General Rules of Subclause 9.1, "Retrieval assignment", are applied to TS and the current row as $TARGET$ and $VALUE$, respectively.

b) Otherwise, values from the current row are assigned to their corresponding targets identified by the <fetch target list>. The assignments are made in an implementation-dependent order. Let TV be a target and let SV denote its corresponding value in the current row of CR. The General Rules of Subclause 9.1, "Retrieval assignment", are applied to TV and SV as $TARGET$ and $VALUE$, respectively.

10) If an exception condition occurs during the assignment of a value to a target, then the values of all targets are implementation-dependent and CR remains positioned on the current row.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <fetch statement> shall not contain a <fetch orientation>.

   b) A <fetch statement> shall not specify FROM.

   c) If the data type of the target identified by the *i*-th <target specification> in the <fetch target list> is an exact numeric type, then the data type of the *i*-th column of the table *T* shall be an exact numeric type.

13.4 <close statement>

Function

Close a cursor.

Format

```
<close statement> ::=
      CLOSE <cursor name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *CR* be the cursor identified by the <cursor name> immediately contained in the <close statement>.

2) If cursor *CR* is not in the open state, then an exception condition is raised: *invalid cursor state*.

3) If the cursor mode of the current SQL-transaction is *cascade off*, and if some <fetch statement> in the current set-processing mode session of the current SQL-transaction has not resulted in a <status parameter> indicating a completion condition of *no data*, then an exception condition is raised: *invalid SQL statement*.

4) Cursor *CR* is placed in the closed state and the copy of the <cursor specification> of the <declare cursor> that specified *CR* is destroyed.

5) Let *CC* be the set of constraints that are defined by <assertion definition>s or by <table constraint definition>s of tables that are nodes in the constraint-referential-action graph for *CR*. Let *CCL* be a comma-separated list of the <constraint name>s of the constraints in *CC*. The following <set constraints mode statement> is effectively executed:

   SET *CCL* CONSTRAINTS IMMEDIATE

   Note: *The constraint-referential-action graph is defined in Subclause 13.2, "<open statement>".*

6) If the cursor mode of the current SQL-transaction is *cascade off*, then the cursor mode of the current SQL-transaction is set to *cascade on*.

7) Any triggered actions that were deferred are effectively executed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

13.5 <select statement: single row>

Function

Retrieve values from a specified row of a table.

Format

```
<select statement: single row> ::=
      SELECT [ <set quantifier> ] <select list>
        INTO <select target list>
          <table expression>

<select target list> ::=
      <target specification> [ { <comma> <target specification> }... ]
```

Syntax Rules

1) Case:

   a) If the <select target list> contains a single <target specification> *TS* and the number of elements in the <select target list> is greater than 1, then:

      i) The data type of *TS* shall be a row type.

      ii) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to *TS* and a row type whose *i*-th element data type is the data type of the *i*-th element of the <select list>, as *TARGET* and *VALUE*, respectively.

   b) Otherwise:

      i) The number of elements in the <select list> shall be the same as the number of elements in the <select target list>. The *i*-th <target specification> in the <select target list> corresponds with the *i*-th element of the <select list>.

      ii) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to each corresponding <target specification> and <value expression>, as *TARGET* and *VALUE*, respectively.

2) Let *S* be a <query specification> whose <select list> and <table expression> are those specified in the <select statement: single row> and that specifies the <set quantifier> if it is specified in the <select statement: single row>. *S* shall be a valid <query specification>.

Access Rules

None.

General Rules

1) Let *Q* be the result of <query specification> *S*.

2) Case:

   a) If the cardinality of *Q* is greater than 1, then an exception condition is raised: *cardinality violation*. It is implementation-dependent whether or not SQL-data values are assigned to the targets identified by the <select target list>.

   b) If *Q* is empty, then no SQL-data values are assigned to any targets identified by the <select target list>, and a completion condition is raised: *no data*.

   c) Otherwise, values in the row of *Q* are assigned to their corresponding targets.

3) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

4) Case:

   a) If the <select target list> contains a single <target specification> *TS* and the number of elements in the <select list> is greater than 1, then the current row is assigned to *TS* and the General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TS* and a row whose *i*-th element is the value of the *i*-th element of the <select list>, as *TARGET* and *VALUE* respectively.

   b) Otherwise:

      i) The assignment of values to targets in the <select target list> is in an implementation-dependent order.

      ii) The target identified by the *i*-th <target specification> of the <select target list> corresponds to the *i*-th value in the row of *Q*.

      iii) Let *TV* be an identified target and let *SV* be its corresponding value in the row of *Q*.

      iv) The General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TV* and *SV*, as *TARGET* and *VALUE*, respectively.

5) If an exception condition is raised during the assignment of a value to a target, then the values of all targets are implementation-dependent.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the data type of the target identified by the *i*-th <target specification> in the <select target list> is an exact numeric type, then the data type of the *i*-th column of the table *T* shall be an exact numeric type.

   b) The <table expression> shall not include a <group by clause> or a <having clause> and shall not identify a grouped view.

13.6 <delete statement: positioned>

Function

Delete a row of a table.

Format

```
<delete statement: positioned> ::=

    ANSI Only---caused by ISO changes not yet considered by ANSI

    DELETE [ FROM <table name> ]

    ISO Only--SQL3

    DELETE [ FROM <table reference> ]

      WHERE CURRENT OF <cursor name>
```

Syntax Rules

1) Let *CR* be the <declare cursor> denoted by the <cursor name>.

2) Either:

   a) There shall be a trigger associated with the simply underlying table of *CR* whose trigger event is DELETE and whose trigger action time is INSTEAD OF; or

   b) *T* shall be an inherently updatable table.

   Note: "inherently updatable cursor" is defined in Subclause 13.1, "<declare cursor>".

   ANSI Only—caused by ISO changes not yet considered by ANSI

3) Let *T* be the simply underlying table of *CR*. *T* is the *subject table* of the <delete statement: positioned>.

4) If <table name> is specified, then:

   a) Let *SUT* be the table identified by <table name>.

   b) *SUT* shall be the only simply underlying table of the only simply underlying table of *T*.

   c) The schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *SUT*.

   ISO Only–SQL3

5) Let *QS* be the <query specification> that is the simply underlying table of the simply underlying table of *CR*. Let *T* be the simply underlying table of *QS*.

6) If <table reference> is specified, then the <table reference> shall identify *T*.

Access Rules

ISO Only–SQL3

1) The applicable privileges for each <table name> in the first <from clause> in the <cursor specification> of *CR* shall include DELETE. If *T* is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT and the <referential action> in the <referential constraint definition> specifies SET NULL or SET DEFAULT, then the applicable privileges for all columns in the <referencing columns> in that <referential constraint definition> shall include UPDATE.

ANSI Only—caused by ISO changes not yet considered by ANSI

2) If there is no trigger associated with the subject table whose <trigger event> is DELETE and whose <trigger action time> is INSTEAD of, then:

   a) The applicable privileges for each <table name> in the first <from clause> in the <cursor specification> of *CR* shall include DELETE.

   b) If the subject table is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT and the <referential action> in the <referential constraint definition> specifies SET NULL or SET DEFAULT, then the applicable privileges for all columns in the <referencing columns> in that <referential constraint definition> shall include UPDATE.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.4, "<privileges>".

General Rules

ISO Only–SQL3

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

2) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

5) If cursor *CR* is not positioned on a row, then an exception condition is raised: *invalid cursor state*.

6) If *CR* is a holdable-cursor and a <fetch statement> has not been issued against *CR* within the current SQL-transaction, then an exception condition is raised: *invalid cursor state*.

7) If the cursor mode of the current SQL-transaction is *cascade off*, and if a prior <SQL procedure statement> executed within the current set-processing mode session of the current SQL-transaction was an <update statement: positioned>, then an exception condition is raised: *invalid SQL statement*.

ANSI Only—caused by ISO changes not yet considered by ANSI

8) If there are any triggers associated with the subject table whose <trigger event> is DELETE and whose <trigger action time> is BEFORE, then execute the <triggered action> of each such trigger.

   Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

9) Case:

   a) If there is a trigger associated with the subject table whose <trigger event> is DELETE and whose <trigger action time> is INSTEAD OF, then execute the <triggered action> of that triggers.

      Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

   b) Otherwise:

      i) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only, and not every leaf generally underlying table of *CR* is a temporary table, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

      ii) The current row of *CR* is effectively deleted by applying the rules for deletion from a derived table.

      Note: See the General Rules of Subclause 7.13, "<query specification>", for deletion from a derived table.

iii) If any leaf generally underlying table of *CR* belongs to a subtable family, then for every row in such a table that is marked for deletion, every row in the corresponding row subtable family is marked for deletion.

10) If there any triggers associated with the subject table whose <trigger event> is DELETE and whose <trigger action time> is AFTER, then execute the <triggered action> of each such trigger.

---

11) If, while *CR* is open, the row from which the current row of *CR* is derived has been marked for deletion by any <delete statement: searched>, marked for deletion by any <delete statement: positioned> that identifies any cursor other than *CR*, updated by any <update statement: searched>, or updated by any <update statement: positioned> that identifies any cursor other than *CR*, then a completion condition is raised: *warning—cursor operation conflict*.

12) Case:

a) If the cursor mode of the current SQL-transaction is *cascade on*, then all rows that are marked for deletion are effectively deleted at the end of the <delete statement: positioned> prior to the checking of any integrity constraint.

b) If the cursor mode of the current SQL-transaction is *cascade off*, then the deletion of all rows that are marked for deletion is effectively deferred until implicitly enacted by the execution of a <commit statement> or explicitly enacted by execution of a <close statement>.

13) If the <delete statement: positioned> deleted the last row of *CR*, then the position of *CR* is after the last row; otherwise, the position of *CR* is before the next row.

14) If the <delete statement: positioned> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

ISO Only–SQL3

---

15) If *T* is a supertable, then every row of the direct and the indirect subtables of *T* in the same subtable row family is also marked for deletion.

- 1 Rule deleted

****Editor's Note****
There is "marking for deletion", but no actual deletion in the General Rules of this Subclause. This has been identified as a possible problem (see Possible Problem **415** in the Editors Notes).

****Editor's Note****
Shouldn't there be General Rules to invoke default destructors for ADT columns? This has been identified as a possible problem (see Possible Problem **416** in the Editors Notes).

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

13.7 <delete statement: searched>

Function

Delete rows of a table.

Format

```
<delete statement: searched> ::=
      DELETE FROM <table reference>
        [ WHERE <search condition> ]
```

Syntax Rules

1) Let *T* be the table identified by the <table reference>. Either:

   a) There shall be a trigger associated with *T* whose trigger event is DELETE and whose trigger action time is INSTEAD OF; or

   b) *T* shall be an inherently updatable table, and *T* shall not be a table derived from a <query expression> that specified INTERSECT or EXCEPT.

2) The scope of the <table reference> is the entire <delete statement: searched>.

3) If the <delete statement: searched> is contained in a <triggered SQL statement>, then the <search condition> shall not contain a <value specification> that specifies a parameter reference.

ISO Only–SQL3

4) The schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*.

ANSI Only—caused by ISO changes not yet considered by ANSI

5) *T* is the *subject table* of the <delete statement: searched>.

6) If WHERE <search condition> is not specified, then WHERE TRUE is implicit.

Access Rules

ISO Only–SQL3

1) The applicable privileges for the <table name> in the <table reference> that is not in a <subquery> shall include DELETE unless the <delete statement: searched> is contained in a <triggered SQL statement>. If *T* is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT and the <referential action> in the <referential constraint definition> specifies SET NULL or SET DEFAULT, then the applicable privileges for all columns in the <referencing columns> in that <referential constraint definition> shall include UPDATE.

ANSI Only—caused by ISO changes not yet considered by ANSI

2) Case:

   a) If <table reference> immediately contains a <table name> *TN*, then the applicable privileges for *TN* shall include DELETE, unless the <delete statement: searched> is contained in a <triggered SQL statement>.

   b) Otherwise, if there is no trigger associated with *T* whose <trigger event> is DELETE and whose <trigger action time> is INSTEAD OF, then:

      i) The applicable privileges for each <table name> in the <table reference> that is not in a <subquery> shall include DELETE unless the <delete statement: searched> is contained in a <triggered SQL statement>.

      ii) Proposal X3H2-94-092/SOU-072 had no text for this subrule.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.4, "<privileges>".

General Rules

ISO Only–SQL3

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

2) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

ISO Only–SQL3

5) Case:

   a) If <search condition> is not specified, then all rows of *T* are marked for deletion.

- 1 sentence deleted

   b) If <search condition> is specified, then it is applied to each row of *T* with the <table reference> bound to that row, and all rows for which the result of the <search condition> is true are marked for deletion.

- 1 sentence deleted

   The <search condition> is effectively evaluated for each row of *T* before marking for deletion any row of *T*.

   Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T*, the reference is to the value of that column in the given row of *T*.

   Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

ANSI Only—caused by ISO changes not yet considered by ANSI

6) If there are any triggers associated with the simply underlying table of *CR* whose <trigger event> is DELETE and whose <trigger action time> is BEFORE, then execute the <triggered action> of each such trigger.

   Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

7) Case:

   a) If there is a trigger associated with *T* whose <trigger event> is DELETE and whose <trigger action time> is INSTEAD OF, then execute the <triggered action> of that triggers.

      Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

   b) Otherwise:

      i) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only, and *T* is neither a temporary table nor a derived table all of whose leaf generally underlying tables are temporary tables, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

ii) The <search condition> is applied to each row of *T* with the <table reference> bound to that row. All rows for which the result of the <search condition> is true are marked for deletion if *T* is a base table; otherwise they are effectively deleted by applying the rules for deletion from a derived table.

Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

iii) The <search condition> is effectively evaluated for each row of *T* before marking for deletion any row of *T* or of any leaf generally underlying table of *T*.

iv) Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results are used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T*, then the reference is to the value of that column in the given row of *T*.

Note: "Outer reference" is defined in Subclause 6.3, "<item reference>".

v) If *T* or any leaf generally underlying table of *T* belongs to a subtable family, then for every row in such a table that is marked for deletion, every tow in the corresponding row subtable family is marked for deletion.

8) If there any triggers associated with the subject table whose <trigger event> is DELETE and whose <trigger action time> is AFTER, then execute the <triggered action> of each such trigger.

---

9) If any row that is marked for deletion by the <delete statement: searched> has been marked for deletion by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open, then a completion condition is raised: *warning—cursor operation conflict*.

10) All rows that are marked for deletion are effectively deleted at the end of the <delete statement: searched>, prior to the checking of any integrity constraints.

11) If <search condition> is specified, then the <search condition> is evaluated for each row of *T* prior to the invocation of any <triggered action> caused by the imminent or actual deletion of any row of *T*.

12) If the <delete statement: searched> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

ISO Only–SQL3

---

13) If *T* is a supertable, then every row of the direct and the indirect subtables of *T* in the same subtable row family is also marked for deletion.

- 1 Rule deleted

---

14) If no row is deleted, then a completion condition is raised: *no data*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in the <search condition>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

13.8 <insert statement>

Function

Create new rows in a table.

Format

```
<insert statement> ::=
       INSERT INTO { <table reference> | CURSOR <cursor name> }
         <insert columns and source>
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
         [ <insert point> ]
```

ANSI Only--SQL3

```
         [ INTO <target specification> ]
```

```
<insert columns and source> ::=
         [ <left paren> <insert column list> <right paren> ] <query expression>
       | DEFAULT VALUES

<insert column list> ::= <column name list>
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
<insert point> ::=
       <relative insert point> ELEMENT <where clause>

<relative insert point> ::=
         BEFORE
       | AFTER
```

Syntax Rules

ANSI Only—caused by ISO changes not yet considered by ANSI

1) If <cursor name> is specified, then <insert point> shall not be specified and the containing <module> shall contain a <declare cursor> *CR* whose <cursor name> is the same as the <cursor name> in the <insert statement>.

2) Case:

   a) If <cursor name> is specified, then let *T* be the underlying table of the <cursor specification>.

   b) Otherwise, let *T* be the table identified by the <table reference>.

3) A column identified by the <insert column list> is an object column.

4) Either:

   a) There shall be a trigger associated with *T* whose trigger event is INSERT and whose trigger action time is INSTEAD OF; or

   b) *T* shall be an inherently updatable table that is not identified in any <from clause> of the <query expression>,
   ANSI each object column of *T* shall be an inherently updatable or constant column,
   and *T* shall not be a table derived from a <query expression> that specifies a set operator UNION or EXCEPT.

   ANSI Only—caused by ISO changes not yet considered by ANSI

5) *T* is the *subject table* of the <insert statement>.

ANSI Only—caused by ISO changes not yet considered by ANSI

6) If <insert point> is specified, then *T* shall be a list table.

7) An <insert columns and source> that specifies DEFAULT VALUES is
   ANSI implicitly replaced by
   ISO equivalent to
   an <insert columns and source> that specifies a <query expression> of the form

   VALUES (DEFAULT, DEFAULT, . . . , DEFAULT)

   where the number of "DEFAULT" entries is equal to the number of columns of *T*.

8) Each <column name> in the <insert column list> shall identify a column of *T*. No <column name> of *T* shall be identified more than once. If
   ISO <insert columns and source> is specified and
   the <insert column list> is omitted, then an <insert column list> that identifies all columns of *T* in the ascending sequence of their ordinal positions within *T* is implicit.

   ANSI Only–SQL3

   <column name> shall not specify the implicit identity column.

9) If <target specification> is specified, then <insert point> shall not be specified and *T* shall have a row identifier defined.

10) If the <insert statement> is not contained in a <triggered SQL statement>, then

Case:

a) If an <insert column list> is specified, then the applicable privileges for each <table name> contained in the <table reference> that is not in a <subquery> shall include INSERT for each <column name> in the <insert column list>.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.4, "<privileges>".

b) If *T* is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT in the <join condition> and LEFT, RIGHT, or FULL in the <join type>, then the applicable privileges for the table identified by that <joined table> shall include UPDATE for all columns in the <referencing columns> in the <referential constraint definition>.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.4, "<privileges>".

- 1 Rule moved earlier in this Subclause

ISO Only–SQL3

11) An object column shall be an inherently updatable or constant column.

12) Let *QT* be the table specified by the <query expression>. The degree of *QT* shall be equal to the number of <column name>s in the <insert column list>. The column of table *T* identified by the *i*-th <column name> in the <insert column list> corresponds with the *i*-th column of *QT*.

13) The Syntax Rules of Subclause 9.2, "Store assignment", apply to corresponding columns of *T* and *QT* as *TARGET* and *VALUE*, respectively.

14) If the <insert statement> is contained in a <triggered SQL statement>, then the insert value shall not contain a <value specification> that specifies a parameter reference.

ISO Only–SQL3

15) The schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*.

Access Rules

1) Case:

ISO Only–SQL3

a) If an <insert column list> is specified, then the applicable privileges shall include INSERT for each <column name> in the <insert column list>.

b) Otherwise, the applicable privileges shall include INSERT for each <column name> in *T*.

ANSI Only—caused by ISO changes not yet considered by ANSI

c) If <table reference> immediately contains a <table name> *TN*, then the applicable privileges for *TN* shall include INSERT for each object column, unless the <insert statement> is contained in a <triggered SQL statement>.

d) Otherwise, if there is not trigger associated with *T* whose <trigger event> is INSERT and whose <trigger action time> is INSTEAD OF, then the applicable privileges shall include INSERT for each counterpart of an object column in an leaf generally underlying table of *T*, unless the <insert statement> is contained in a <triggered SQL statement>.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.4, "<privileges>".

General Rules

ISO Only–SQL3

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

2) If there is any open cursor *CR* in the current SQL-transaction that was opened CASCADE OFF and whose constraint-referential-action graph is not distinct from the constraint-referential-action graph of the <insert statement>, then an exception condition is raised: *invalid SQL statement*.

Note: *The constraint-referential-action graph is defined in Subclause 13.2, "<open statement>".*

3) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

5) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

ANSI Only—caused by ISO changes not yet considered by ANSI

6) If *T* is a list table and <cursor name> is not specified, then:

   Case:

   a) If <insert point> is specified, then:

      Case:

      i) If the result of the <search condition> immediately contained in the <where clause> is true for one and only one row of *T*, then any rows to be inserted BEFORE that row will be inserted in order with the last insertion immediately preceding that row; any rows to be inserted AFTER that row will be inserted in order with the first insertion immediately following that row.

      ii) Otherwise, an exception condition is raised: *data exception—insertion point not unique*.

   b) Otherwise, any rows to be inserted will be inserted in order following the last row of the table, if any, or beginning at position 1 if the table is empty.

ISO Only–SQL3

7) If <insert columns and source> is specified and immediately contains a <query expression>, then let *Q* be the result of that <query expression>. If *Q* is empty, then no row shall be inserted and a completion condition is raised: *no data*.

8) Let *B* be the leaf generally underlying table of *T*.

****Editor's Note****

Possible Problem **250** in the Editor's Notes identifies a concern over the use of "leaf generally underlying table" here; in SQL-92, an updatable view could have only one leaf generally underlying table, but in SQL3, more views are updatable and there may be more than one leaf generally underlying table involved.

9) The <query expression> is effectively evaluated before inserting any rows into *B*.

10) Let *Q* be the result of that <query expression>.

    Case:

    a) If *Q* is empty, then no row is inserted and a completion condition is raised: *no data*.

b) Otherwise, for each row *R* of *Q*:

i) A candidate row of *B* is effectively created in which the value of each column is its default value, as specified in the General Rules of Subclause 11.9, "<default clause>". The candidate row includes every column of *B*.

ii) If <insert columns and source> immediately contains a <query expression>, then each object column in the candidate row is updated and set equal to the corresponding insert value.

iii) For every object column in the candidate row, the value of the object column identified by the *i*-th <column name> in the <insert column list> is replaced by the *i*-th value of *R*.

ANSI Only–SQL3

---

iv) For each column in the candidate row whose data type is an object abstract data type and whose declaration specifies INSTANCE, an object identifier is generated and associated with that column.

ISO Only—caused by ANSI changes not yet considered by ISO

---

v) If the candidate row has a column named OID whose value is a null value, then assign to it a unique object identifier value.

---

vi) Let *C* be a column that is represented in the candidate row and let *SV* be its value in the candidate row. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

vii) If *T* is a list
ANSI table
and <cursor name> is specified, then

1) If *CR* is positioned on a row, then the candidate row is inserted after that row and *CR* is positioned on the new row.

2) If *CR* is positioned before a row or after the last row, then the candidate row is inserted at that position and *CR* is positioned on the new row.

viii) Case:

1) If *T* is a set
ANSI table
and the candidate row is equal to an existing row in the table, then a completion condition is raised: *warning—duplicate element eliminated in set insertion*.

2) Otherwise, the candidate row is inserted into the corresponding stored columns of *B*.

Note: The data values allowable in the candidate row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.26, "<view definition>".

ANSI Only—caused by ISO changes not yet considered by ANSI

11) The <query expression> immediately contained in <insert columns and source> is effectively evaluated before insertion of any rows into *T*.

12) Let *Q* be the result of that <query expression>.

Case:

a) If *Q* is empty, then no row is inserted and a completion condition is raised: *no data*. No further General Rules of this Subclause are applied.

b) Otherwise, for each row *R* of *Q*:

i) A candidate row of *T* is effectively created in which the value of each column is its default value, as specified in the General Rules of Subclause 11.9, "<default clause>". The candidate row includes every column of *T*.

ii) For each object column in the candidate row, let *C* be the object column identified iby the *i*-th <column name> in the <insert column list> and let *SV* be the *i*-th value of *R*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *SOURCE*, respectively.

Note: The data values allowable in the candidate row may be constrainted by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.26, "<view definition>".

13) If there are any triggers associated with *T* whose <trigger event> is INSERT and whose <trigger action time> is BEFORE, then execute the <triggered action> of each such trigger.

14) Case:

a) If there is a trigger associated with *T* whose <trigger event> is INSERT and whose <trigger action time> is INSTEAD OF, then execute the <triggered action> of that triggers.

Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

b) Otherwise:

i) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only, and *T* is neither a temporary table nor a derived table all of whose leaf generally underlying tables are temporary tables, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

ii) For each candidate row defined above, in the order of the corresponding rows of *Q* if *Q* is a
ANSI list table:
ISO list:

1) If *T* is a list
ANSI table
and <cursor name> is specified, then:

A) If *CR* is positioned on a row, then the candidate row is inserted after that row and *CR* is positioned on the new row.

B) If *CR* is positioned before a row or after the last row, then the candidate row is inserted at that position and *CR* is positioned on the new row.

2) Case:

A) If *T* is a base table, then:

I) If *T* is a set
ANSI table
and the candidate row is equal to an existing row in the rable, then a completion condition is raised: *warning—duplicate element eliminated in set insertion*.

II) Otherwise, the candidate row is inserted into *T*.

B) Otherwise, the candidate row is effectively inserted by applying the rules for insertion into a derived table.

Note: See the General Rules of Subclause 7.13, "<query specification>" for insertion into a derived table.

15) If there any triggers associated with the subject table whose <trigger event> is INSERT and whose <trigger action time> is AFTER, then execute the <triggered action> of each such trigger.

---

16) If the <insert statement> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

17) If *T* is a subtable, then for each supertable T_i of the subtable family of *T*:

a) If there is a row of T_i that has the same primary key value as the value specified in the <insert column list> or the <query expression> of the <insert statement>, and if any values of that row of T_i are different from the values of the corresponding columns for the inherited columns specified in the <insert column list> or the <query expression> of the <insert statement>, then an exception condition is raised: *data exception—invalid update value*.

b) If there is no row of T_i that has the same primary key value as the value specified in the <insert column list> or the <query expression> of the <insert statement>, then a row is inserted into T_i. Values of the new row in T_i ar the same as the values of the inherited columns of the new row of *T*.

Note: If *T* is a maximal supertable, then a row is inserted only into *T*.

ANSI Only–SQL3

18) If *T* has a row identifier defined, then let *TN* be the name of *T*; a new globally unique row identifier value *I* is assigned to the implicit column IDENTITY of *T* and of every supertable of *T*. The type of *I* is the <row identifier type> specified by "*TN* IDENTITY". *I* is assigned to the <target specification>.

Leveling Rules

1) The following restrictions apply for Full SQL:

ANSI Only—caused by ISO changes not yet considered by ANSI

a) An <insert statement> shall not specify an <insert point> or a <cursor name>.

ISO Only—caused by ANSI changes not yet considered by ISO

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) The leaf generally underlying table of *T* shall not be generally contained in the <query expression> immediately contained in the <insert columns and source> except as the <item qualifier> of a column reference.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) The <query expression> that is contained in an <insert columns and source> shall be a <query specification> or it shall be a <table value constructor> that contains exactly one <row value constructor> of the form "<left paren> <row value constructor list> <right paren>", and each <row value constructor element> of that <row value constructor list> shall be a <value specification> or a <null specification>.

b) If the data type of the target identified by the *i*-th <column name> is an exact numeric type, then the data type of the *i*-th item of the <insert statement> shall be an exact numeric type.

c) If the data type of the target *C* identified by the *i*-th <column name> is character string, then the length in characters of the *i*-th item of the <insert statement> shall be less than or equal to the length of *C*.

d) The <insert columns and source> shall immediately contain a <query expression>.

13.9 <update statement: positioned>

Function

Update a row of a table.

Format

```
<update statement: positioned> ::=

        ISO Only--SQL3

        UPDATE [ <table reference> ]

        ANSI Only---caused by ISO changes not yet considered by ANSI

        UPDATE [ <table name> ]

          SET [ <update type> ] <set clause list>
            WHERE CURRENT OF <cursor name>

<set clause list> ::=
       <set clause> [ { <comma> <set clause> }... ]

<set clause> ::=
       <update target> <equals operator> <row value expression>

<update target> ::=
         <object column>
       | <left paren> <object column list> <right paren>
       | ROW

<object column list> ::=
       <object column> [ { <comma> <object column> }... ]

<update type> ::= ALL | SOME | NONE

<object column> ::= <column name>
```

Syntax Rules

1) If <update type> is not specified, then an <update type> of ALL is implicit.

2) Let *CR* be the <declare cursor> denoted by the <cursor name>.

   ISO Only–SQL3

3) Either:

   a) There shall be one or more triggers associated with the simply underlying table of *CR* whose trigger event is UPDATE (spanning all columns of that table identified as <object column>s) and whose trigger action time is INSTEAD OF; or

b) *CR* shall be an inherently updatable cursor.

ANSI Only—caused by ISO changes not yet considered by ANSI

4) Let *T* be the simply underlying table of *CR*. *T* is the *subject table* of the <update statement: positioned>.

5) Either:

a) There shall be one or more triggers associated with *T* whose <trigger event> is UPDATE and whose <trigger action time> is INSTEAD OF, where each <object column> contained in the <set clause list> is identified in one of their <trigger column list>s, or

b) *T* shall be an inherently updatable table.

6) If <table name> is specified, then:

a) Let *SUT* be the table identified by <table name>.

b) *SUT* shall be the only simply underlying table of the only simply underlying table of *T*.

c) The schema identified by the explicit or implicit qualifier of <table name> shall include the descriptor of *SUT*.

Note: "inherently updatable cursor" is defined in Subclause 13.1, "<declare cursor>".

7) Let *T* be the table identified by the <table reference>. Let *QS* be the <query specification> that is the simply underlying table of the simply underlying table of *CR*. The simply underlying table of *QS* shall be *T*.

Note: The "simply underlying table" of a <cursor specification> is defined in Subclause 13.1, "<declare cursor>".

8) If an <update target> specifies ROW, then:

a) There shall be exactly one <set clause> *SC*.

b) *SC* shall not contain an <update type>.

c) The Syntax Rules of Subclause 9.2, "Store assignment", apply to the row of *T* as *TARGET* and the <row value expression> of *SC* as *VALUE*, respectively.

9) Each <column name> specified as an <object column> shall identify a column of *T*.

ANSI Only–SQL3

<object column> shall not specify the implicit identity column.

10) If *CR* is an ordered cursor, then for each <object column> *OC*, the column of *T* identified by *OC* shall not be directly or indirectly referenced in the <order by clause> of the defining <cursor specification> for *CR*.

11) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in any <value expression> immediately contained in any <update source> contained in the <set clause list>.

12) A <value expression> in a <set clause> shall not directly contain a <set function specification>.

13) The same <object column> shall not appear more than once in a <set clause list>.

14) If the cursor identified by <cursor name> was specified using an explicit or implicit <updatability clause> of FOR UPDATE, then each <column name> specified as an <object column> shall identify a column in the explicit or implicit <column name list> associated with the <updatability clause>.

15) If *T* is derived from a <joined table> that specifies FOREIGN KEY or CONSTRAINT, then each <column name> specified as an <object column> shall not identify a column of a referenced table of that <joined table>.

16) The scope of the <table reference> is the entire <update statement: positioned>.

17) For every <object column> in a <set clause>, the Syntax Rules of Subclause 9.2, "Store assignment", apply to the column of *T* identified by the <object column> and the <row value expression> of the <set clause> as *TARGET* and *VALUE*, respectively.

18) A <null state> shall identify a null state in the null class of the column identified by <object column>.

ISO Only—caused by ANSI changes not yet considered by ISO

19) The <set clause> shall not contain a <column name> of OID.

ISO Only–SQL3

20) The schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*.

Access Rules

ISO Only–SQL3

1) The applicable privileges shall include UPDATE for each <object column>.

2) If *T* is derived from a <joined table> that specifies LEFT, RIGHT, or FULL, then the applicable privileges for each <table name> in that <joined table> that is not in a <subquery> shall include INSERT.

ANSI Only—caused by ISO changes not yet considered by ANSI

3) If there are not triggers associated with *T* whose <trigger event> is UPDATE and whose <trigger action time> is INSTEAD OF, where each <object column> contained in the <set clause list> is identified in one of their <trigger column list>s, then:

   a) The applicable privileges shall include UPDATE for each <object column>.

   b) If the subject table is a <joined table> that specifies LEFT, RIGHT, or FULL, then the applicable privileges for each <table name> in that <joined table> that is not in a <subquery> shall include INSERT.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.4, "<privileges>".

General Rules

1) If the <open statement> that opened *CR* specified CASCADE OFF, then the values of USER and any <datetime value function> generally contained in <set clause> shall be those that would have resulted at the time of the execution of the <open statement> which opened *CR*.

ISO Only–SQL3

2) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

5) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

6) If the cursor mode of the current SQL-transaction is *cascade off*, then if the <update statement: positioned> is different from any <update statement: positioned> that had previously been executed in the current set-processing mode session of the current SQL-transaction, or if a <delete statement: positioned> had previously been executed in the current set-processing mode session of the current SQL-transaction, then an exception condition is raised: *invalid SQL statement*.

7) If the cursor mode of the current SQL-transaction is *cascade off*, then if the value of some <simple target specification> that is contained in the <set clause> has changed its value since the last time an <update statement: positioned> was last executed in the current set-processing mode session of the current SQL-transaction, then an exception condition is raised: *invalid target specification value*.

8) If the cursor mode of the current SQL-transaction is *cascade off*, and if the previous <SQL procedure statement> executed within the current set-processing mode session of the current SQL-transaction was an <update statement: positioned>, then an exception condition is raised: *invalid SQL statement*.

9) If cursor *CR* is not positioned on a row, then an exception condition is raised: *invalid cursor state*.

10) If *CR* is a holdable-cursor and a <fetch statement> has not been issued against *CR* within the current SQL-transaction, then an exception condition is raised: *invalid cursor state*.

ISO Only–SQL3

11) The object row is that row from which the current row of *CR* is derived.

ANSI Only—caused by ISO changes not yet considered by ANSI

12) An object row is any row of a base table from which the current row of *CR* is derived.

13) If, while *CR* is open,
ANSI an
ISO the
object row has been marked for deletion by any <delete statement: searched>, marked for deletion by any <delete statement: positioned> that identifies any cursor other than *CR*, updated by any <update statement: searched>, or updated by any <update statement: positioned> that identifies any cursor other than *CR*, then a completion condition is raised: *warning—cursor operation conflict*.

14) The value associated with DEFAULT is the default value for the <object column> in the containing <set clause>, as indicated in the General Rules of Subclause 11.9, "<default clause>".

ISO Only–SQL3

15) The <value expression>s are effectively evaluated before updating the object row. If a <value expression> contains a reference to a column of *T* or to *T* itself, then the reference is to the value of that column or table in the object row before any value of the object row is updated.

16) *CR* remains positioned on its current row, even if an exception condition is raised during derivation of any <value expression> associated with the object row.

17) The update value is the value specified by <row value expression>. If a <query expression> in a <row value expression> is specified and the result of the <query expression> is empty, then

Case:

a) If ALL is specified, then the result of the <row value expression> shall be *D* general null values, where *D* is the degree of the <query expression>.

b) If SOME or NONE is specified, then no object column shall be updated by the result of the <row value expression>.

18) If a <query expression> in a <row value expression> is specified, then:

Case:

a) If ALL or SOME is specified, then for each <set clause>, the value of the specified object column will be replaced by the specified update value.

b) If NONE is specified and the result of the <query expression> is empty, then the object row will not be changed.

19) Case:

a) If a <set clause> specifies an <update target> of ROW, then

Case:

i) If the value *SV* of the <row value expression> is a null value, an exception condition is raised: *data exception—null row not permitted in table*.

ii) Otherwise, the object row is replaced by the value *SV* of the <row value expression>. The General Rules of Subclause 9.2, "Store assignment", are applied to the object row and *SV* as *TARGET* and *VALUE*, respecively.

b) Otherwise:

i) A <set clause> specifies an object column and an update value of that column. The object column is the column identified by the <object column> in the <set clause>. The update value is the value specified by the <update source>.

Note: The data values allowable in the object row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.26, "<view definition>".

ii) The object row is updated as specified by each <set clause>. For each <set clause>, the value of the specified object column, denoted by *C*, is replaced by the specified update value, denoted by *SV*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

ANSI Only—caused by ISO changes not yet considered by ANSI

20) The <row value constructor> is effectively evaluated for each current row before updating any of the current row's object rows.

21) *CR* remains positioned on its current row, even if an exception condition is raised during evaluation of any <row value constructor>.

22) A <set clause> specifies one or more object columns and an update value. An object column is a column identified by an <object column> in the <set clause>. The update value is the value specified by the <row value constructor>.

Note: The data values allowable in the current row may be constrainted by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.26, "<view definition>".

23) If a <set clause> contains a <row value constructor> that immediately contains a <row subquery> *RS*, and the result of *RS* is an empty table, then

Case:

a) If ALL is specified, then the update value for that <set clause> is *D* general null values, where *D* is the degree of *RS*.

b) If SOME is specified, then no object column of that <set clause> is updated by the result of the <row value constructor>.

c) If NONE is specified, then no object row is changed.

24) A candidate new row is constructed by copying the current row of *CR* and updating it as specified by each <set clause>. For each <set clause>, unless otherwise indicated for SOME and NONE in the General Rule above, the value of the column identified by the *i*-th object column in the <set clause>, denoted by *C*, is replaced by the *i*-th column value of the specified update value, denoted by *SV*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

25) If there is any trigger associated with the simply underlying table of *CR* whose <trigger event> is UPDATE and whose <trigger action time> is BEFORE, and that identifies an object column in its <trigger column list>, then execute the <triggered action> of each such trigger.

Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

26) Case:

a) If there are any triggers associated with *T* whose <trigger event> is UPDATE and whose <trigger action time> is INSTEAD OF, where each object column is identified in one of their <trigger column list>s, then execute the <triggered action> of each such trigger.

Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

b) Otherwise:

i) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only and not every leaf generally underlying table of *CR* is a temporary table, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

ii) The current row of *CR* is effectively updated by applying the rules for updating a derived table using the candidate new row.

Note: See the General Rules of Subclause 7.13, "<query specification>", for updating a derived table.

iii) If any leaf generally underlying table of *T* belongs to a subtable family, then for every row in such a table that is marked for deletion, every row in the corresponding row subtable family is marked for deletion.

27) If there is any trigger associated with the simply underlying table of *CR* whose <trigger event> is UPDATE and whose <trigger action time> is AFTER, and that identifies an object column in its <trigger column list>, then execute the <triggered action> of each such trigger.

Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

---

28) If the <update statement: positioned> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

ANSI Only—caused by ISO changes not yet considered by ANSI

---

29) Let *LGUT* be any leaf generally underlying table of *CR* that has been updated by these General Rules.

---

30) If
ANSI *LGUT*
ISO *T*
is a supertable, then every row of the direct and the indirect subtables of
ANSI *LGUT*
ISO *T*
in the same subtable row family is changed such that all column values
ANSI inheritable
ISO inherited
from
ANSI *LGUT*
ISO *T*
replicate those of
ANSI *LGUT*.
ISO *T*.

31) If
ANSI *LGUT*
ISO *T*
is a subtable and the <column name> of the <object column> specifies an inherited column, then every row in the corresponding subtable row family is changed such that all column values inherited by
ANSI *LGUT*
ISO *T*
replicate those of
ANSI *LGUT*.
ISO *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <update statement: positioned> shall contain a <table name>.

   b) An <update statement: positioned> shall not contain an <update type>.

   c) An <update target> shall not contain an <object column list>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) *CR* shall not be an ordered cursor.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the data type of the column identified by the *i*-th <object column> is an exact numeric type, then the data type of the *i*-th <value expression> in the <update statement: positioned> shall be an exact numeric type.

   b) If the data type of the column identified by the *i*-th <object column> *C* is character string, then the length in characters of the *i*-th <value expression> in the <update statement: positioned> shall be less than or equal to the length of *C*.

   c) An <update source> shall not specify DEFAULT.

13.10 <update statement: searched>

Function

Update rows of a table.

Format

```
<update statement: searched> ::=
       UPDATE <table reference>
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
         <update mechanism>
```

ISO Only---caused by ANSI changes not yet considered by ISO

```
         SET [ <update type> ] <set clause list>
```

```
         [ WHERE <search condition> ]
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
<update mechanism> ::=
         <update by setting>
       | <update by moving>

<update by setting> ::=
       SET [ <update type> ] <set clause list>

<update by moving> ::=
       MOVE <insert point>
```

Syntax Rules

1) If <update type> is not specified, then an <update type> of ALL is implicit.

ANSI Only—caused by ISO changes not yet considered by ANSI

2) If <update by moving> is specified, then *T* shall be a base table that is a list table and shall not be a supertable or a subtable.

ISO Only–SQL3

3) Let *T* be the table identified by the <table reference>.

ANSI Only—caused by ISO changes not yet considered by ANSI

4) Let *T* be the table identified by the <table reference>. Either:

   a) <update by setting> is specified and there shall be one or more triggers associated with *T* whose <trigger event> is UPDATE and whose <trigger action time> is INSTEAD OF, where each <object column> contained in the <set clause list> is identified in one of their <trigger column list>s, or

   b) *T* shall be an inherentily updatable table.

5) *T* is the *subject table* of the <update statement: searched>.

6) Either:

   a) There shall be one or more triggers associated with *T* whose trigger event is UPDATE (spanning all columns of that table identified as <object column>s) and whose trigger action time is INSTEAD OF; or

   b) *T* shall be an inherently updatable table.

7) If an <update target> specifies ROW, then:

   a) There shall be exactly one <set clause> *SC*.

   b) *SC* shall not contain an <update type>.

   c) The Syntax Rules of Subclause 9.2, "Store assignment", apply to the row of *T* as *TARGET* and the <row value expression> of *SC* as *VALUE*, respectively.

8) A <value expression> in a <set clause> shall not directly contain a <set function specification>.

9) Each <column name> specified as an <object column> shall identify a column of *T*.

ANSI Only–SQL3

<object column> shall not specify the implicit identity column.

10) The same <object column> shall not appear more than once in a <set clause list>.

11) If *T* is derived from a <joined table> which specifies FOREIGN KEY or CONSTRAINT, then no <column name> specified as an <object column> shall identify a column of the referenced table of that <joined table>.

12) The scope of the <table reference> is the entire <update statement: searched>.

13) For every <object column> in a <set clause>, the Syntax Rules of Subclause 9.2, "Store assignment", apply to the column of *T* identified by the <object column> and the <value expression> of the <set clause> as *TARGET* and *VALUE*, respectively.

14) A <null state> shall identify a null state in the null class of the column identified by <object column>.

15) If the <update statement: searched> is contained in a <triggered SQL statement>, then the <search condition> shall not contain a <value specification> that specifies a parameter reference.

ISO Only–SQL3

16) The schema identified by the explicit or implicit qualifier of the <table name> shall include the descriptor of *T*.

Access Rules

ISO Only–SQL3

1) The applicable privileges for each <table name> in <table reference> shall include UPDATE for each <object column> unless the <update statement: searched> is contained in a <triggered SQL statement>. If *T* is derived from a <joined table> that specifies LEFT, RIGHT, or FULL, then the applicable privileges for each <table name> in that <joined table> that is not in a <subquery> shall include INSERT for all columns unless the <update statement: searched> is contained in a <triggered SQL statement>.

ANSI Only—caused by ISO changes not yet considered by ANSI

2) Case:

   a) If <table reference> immediately contains a <table name> *TN*, then the applicable privileges for *TN* shall include UPDATE for each <object column>, unless the <update statement: searched> is contained in a <triggered SQL statement>.

   b) Otherwise, if there are not triggers associated with *T* whose <trigger event> is UPDATE and whose <trigger action time> is INSTEAD OF, where each <object column> contained in the <set clause list> is identified in one of their <trigger column list>s, then:

      i) The applicable privilges for each <table name> in <table reference> that is not in a <subquery> shall include UPDATE for each <object column>, unless the <update statement: searched> is contained in a <triggered SQL statement>.

      ii) If *T* is derived from a <joined table> that specifies LEFT, RIGHT, or FULL, then the applicable privileges for each <table name> in that <joined table> that is not in a <subquery> shall include INSERT for all columns unless the <update statement: searched> is contained in a <triggered SQL statement>.

Note: The *applicable privileges* for a <table name> are defined in Subclause 10.4, "<privileges>".

General Rules

ISO Only–SQL3

1) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

2) If there is any open cursor *CR* in the current SQL-transaction that was opened CASCADE OFF and whose constraint-referential-action graph is not disjoint from the constraint-referential-action graph of the <update statement: searched>, then an exception condition is raised: *invalid SQL statement*.

****Editor's Note****
The preceding rules deals with CASCADE ON and OFF and depend on the constraint-referential-action graph, which has been deleted. Therefore, these rules will need to be revised.

3) If there is any cursor *CR* that is currently open and whose <declare cursor> specified SENSITIVE, then either the change resulting from the successful execution of this statement shall be made visible to *CR* or an exception condition is raised: *cursor sensitivity exception—request failed*.

4) If there is any cursor *CR* that is currently open and whose <declare cursor> contained INSENSITIVE, then either the change resulting from the successful execution of this statement shall be invisible to *CR*, or an exception condition is raised: *cursor sensitivity exception—request failed*.

5) The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

6) Case:

   a) If a <search condition> is not specified, then all rows of *T* are the
   ANSI subject
   ISO object
   rows.

   b) If a <search condition> is specified, then it is applied to each row of *T* with the <table name> bound to that row, and the
   ANSI subject
   ISO object
   rows are those rows for which the result of the <search condition> is true. The <search condition> is effectively evaluated for each row of *T* before updating any row of *T*.

Each <subquery> in the <search condition> is effectively executed for each row of *T* and the results used in the application of the <search condition> to the given row of *T*. If any executed <subquery> contains an outer reference to a column of *T* or to *T* itself, then the reference is to the value of that column in the given row of *T*.

Note: *Outer reference* is defined in Subclause 6.3, "<item reference>".

ANSI Only—caused by ISO changes not yet considered by ANSI

7) If *T* is a base table, then each subject row is also an *object row*; otherwise, an *object row* is any row of a leaf generally underlying table of *T* from which a subject row is derives.

8) If any row in the set of object rows has been marked for deletion by any <delete statement: positioned> that identifies some cursor *CR* that is still open or updated by any <update statement: positioned> that identifies some cursor *CR* that is still open, then a completion condition is raised: *warning—cursor operation conflict*.

9) If a <search condition> is specified, then the <search condition> is evaluated for each row of *T* prior to the invocation of any <triggered action> caused by the update of any row of *T*.

ISO Only–SQL3

10) The update value is the value specified by <row value expression>. If a <query expression> in a <row value expression> is specified and the result of the <query expression> is empty, then

    Case:

    a) If ALL is specified, then the result of the <row value expression> shall be *D* general null values, where *D* is the degree of the <query expression>.

    b) If SOME or NONE is specified, then no object column shall be updated by the result of the <row value expression>.

11) If a <query expression> in a <row value expression> is specified, then:

    Case:

    a) If ALL or SOME is specified, then for each <set clause>, the value of the specified object column is replaced by the specified update value.

    b) If NONE is specified and the result of the <query expression> is empty, then the object row is not changed.

12) If the set of object rows is empty, then a completion condition is raised: *no data*.

13) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

ISO Only–SQL3

14) The <value expression>s are effectively evaluated for each row of *T* before updating any row of *T*.

15) Case:

a) If a <set clause> specifies an <update target> of ROW, then:

i) If the value *SV* of the <row value expression> is a null value, an exception condition is raised: *data exception—null row not permitted in table*.

ii) Otherwise, the object row is replaced by the value *SV* of the <row value expression>. The General Rules of Subclause 9.2, "Store assignment", are applied to the object row and *SV* as *TARGET* and *VALUE*, respectively.

b) Otherwise:

i) A <set clause> specifies an object column an an update value of that column. The object column is the column identified by the <object column> in the <set clause>. The update value is the value specified by the <update source>.

Note: The data values allowable in the object row may be constrained by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.26, "<view definition>".

ii) The object row is updated as specified by each <set clause>. For each <set clause>, the value of the specified object column, denoted by *C*, is replaced by the specified update value, denoted by *SV*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

ANSI Only—caused by ISO changes not yet considered by ANSI

16) If <update by moving> is specified, then:

a) Case:

i) If the result of the <search condition> immediately contained in the <where clause> immediately contained in the <insert point> is true for one and only one row of *T*, then an ordered set of rows to be moved BEFORE that row will be inserted in order with the last insertion immediately preceding that row; an ordered set of rows to be moved AFTER that row will be inserted in order with the first insertion immediately following that row.

ii) Otherwise, an exception condition is raised: *data exception—insertion point not unique*.

b) No further General Rules of this Subclause are applied.

17) The <row value constructor> of each <set clause> is effectively evaluated for each row of *T* before updating any row of *T*.

18) A <set clause> specifies one or more object columns and an update value. An object column is a column identified by an <object column> in the <set clause>. The update value is the value specified by the <row value constructor>.

Note: The data values allowable in the current row may be constrainted by a WITH CHECK OPTION constraint. The effect of a WITH CHECK OPTION constraint is defined in the General Rules of Subclause 11.26, "<view definition>".

19) If a <set clause> contains a <row value constructor> that immediately contains a <row subquery> *RS*, and the result of *RS* is an empty table, then for each subject row

Case:

a) If ALL is specified, then the update value for that <set clause> is *D* general null values, where *D* is the degree of *RS*.

b) If SOME is specified, then no object column of that <set clause> is updated by the result of the <row value constructor>.

c) If NONE is specified, then no object row is changed.

20) For each subject row, a candidate new row is constructed by copying the subject row and updating it as specified by each <set clause>. For each <set clause>, unless otherwise indicated for SOME and NONE in the General Rule above, the value of the column identified by the *i*-th object column in the <set clause>, denoted by *C*, is replaced by the *i*-th column value of the specified update value, denoted by *SV*. The General Rules of Subclause 9.2, "Store assignment", are applied to *C* and *SV* as *TARGET* and *VALUE*, respectively.

21) If there is any trigger associated with the simply underlying table of *CR* whose <trigger event> is UPDATE and whose <trigger action time> is BEFORE, and that identifies an object column in its <trigger column list>, then execute the <triggered action> of each such trigger.

Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

22) Case:

a) If there are any triggers associated with *T* whose <trigger event> is UPDATE and whose <trigger action time> is INSTEAD OF, where each object column is identified in one of their <trigger column list>s, then execute the <triggered action> of each such trigger.

Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

b) Otherwise:

i) If the access mode of the current SQL-transaction or the access mode of the branch of the current SQL-transaction at the current SQL-connection is read-only and *T* is neither a temporary table nor a derived table all of whose leaf generally underlying tables are temporary tables, then an exception condition is raised: *invalid transaction state—read-only SQL-transaction*.

ii) Case:

1) If *T* is a base table, then each subject row in *T* is replaced by its candidate new row.

2) Otherwise, each subject row is effectively updated by applying the rules for updating a derived table using its candidate new row.

Note: See the General Rules of Subclause 7.13, "<query specification>", for updating a derived table.

23) If there is any trigger associated with the simply underlying table of *CR* whose <trigger event> is UPDATE and whose <trigger action time> is AFTER, and that identifies an object column in its <trigger column list>, then execute the <triggered action> of each such trigger.

Note: See the General Rules of Subclause 11.45, "<trigger definition>", for execution of <triggered action>s.

---

24) If the <update statement: searched> is the triggering statement for an <assertion definition> and the constraint mode of the assertion in the current SQL-transaction is *deferred*, then the <assertion definition> becomes a pending <assertion definition>.

ANSI Only—caused by ISO changes not yet considered by ANSI

---

25) If *T* is a base table, the let *LGUT* be *T*; otherwise, let *LGUT* be any leaf generally underlying table of *T* that has been updated by these General Rules.

---

26) If
ANSI *LGUT*
ISO *T*
is a supertable, then every row of the direct and the indirect subtables of
ANSI *LGUT*
ISO *T*
in the same subtable row family is changed such that all column values inherited from
ANSI *LGUT*
ISO *T*
replicate those of
ANSI *LGUT*.
ISO *T*.

27) If
ANSI *LGUT*
ISO *T*
is a subtable and the <column name> of the <object column> specifies an inherited column, then every row in the corresponding subtable row family is changed such that all column values
ANSI inheritable
ISO inherited
by
ANSI *LGUT*
ISO *T*
replicate those of
ANSI *LGUT*.

ISO *T*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) An <update statement: searched> shall not contain an <update type>.

   b) An <update target> shall not contain an <object column list>.

   ANSI Only—caused by ISO changes not yet considered by ANSI

   c) An <update statement: searched> shall not contain an <update by moving>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in the <search condition> or in any <value expression> immediately contained in any <update source> contained in the <set clause list>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the data type of the column identified by the *i*-th <object column> is an exact numeric type, then the data type of the *i*-th <value expression> in the <update statement: searched> shall be an exact numeric type.

   b) If the data type of the column identified by the *i*-th <object column> *C* is character string, then the length in characters of the *i*-th <value expression> in the <update statement: searched> shall be less than or equal to the length of *C*.

• 2 Subclauses deleted.

13.11 <temporary table declaration>

Function

Declare a declared local temporary table.

Format

```
<temporary table declaration> ::=
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
DECLARE LOCAL TEMPORARY <table type> <table name>
  <table element list>
```

ISO Only--SQL3

```
DECLARE LOCAL TEMPORARY TABLE <table name>
  <table element list>
```

```
  [ ON COMMIT <table commit action> ROWS ]
```

Syntax Rules

1) Let *TN* be the <table name> of a <temporary table declaration> *TTD*, and let *T* be the <qualified identifier> of *TN*.

   Case:

   a) If *TN* contains a <local or schema qualifier> *LSQ*, then *TTD* shall be contained in a <module> *M* and *LSQ* shall be either "MODULE" or the <module name> of *M*.

   b) If *TN* does not contain a <local or schema qualifier>, then *TTD* shall be contained in a <module> *M* and "MODULE" is implicit.

2) If a <temporary table declaration> is contained in a <module>, then

   a) If *TN* contains a <local or schema qualifier>, then it shall be "MODULE".

   b) The <qualified identifier> of *TN* shall be different from the <qualified identifier> of the <table name> of any other <temporary table declaration> or <temporary view declaration> that is contained in *M*.

3) The descriptor of the table defined by a <temporary table declaration> includes the name of *T* and the column descriptor specified by each <column definition>. The *i*-th column descriptor is given by the *i*-th <column definition>.

4) A <temporary table declaration> shall contain at least one <column definition>.

5) If ON COMMIT is not specified, then ON COMMIT DELETE ROWS is implicit.

Access Rules

None.

General Rules

1) Let *U* be the implementation-dependent <schema name> that is effectively derived from the implementation-dependent SQL-session identifier associated with the SQL-session and an implementation-dependent name associated with the <module> that contains the <temporary table declaration>.

2) The definition of *T* within a module is effectively equivalent to the definition of a persistent base table *U*.*T*. Within the module, any reference to MODULE.*T* is equivalent to a reference to *U*.*T*.

3) A set of privilege descriptors is created that define the privileges INSERT, SELECT, UPDATE, DELETE, and REFERENCES on this table and INSERT, SELECT, UPDATE, and REFERENCES for every <column definition> in the table definition to *A*. These privileges are not grantable. The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM".

4) The definition of a temporary table persists for the duration of the SQL-session. The termination of the SQL-session is effectively followed by the execution of the following <drop table statement> with the current <authorization identifier> and current <schema name> *U* without further Access Rule checking:

   DROP TABLE *T* CASCADE

5) The definition of a declared local temporary table does not appear in any view of the Information Schema.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <temporary table declaration>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

13.12 <temporary abstract data type declaration>

Function

Declare a temporary abstract data type.

Format

```
<temporary abstract data type declaration> ::=
       DECLARE TEMPORARY TYPE <abstract data type name>
           <abstract data type body>
```

Syntax Rules

1) Let *ADTN* be the <abstract data type name> of a <temporary abstract data type declaration> *TTD*.

   Case:

   a) If *ADTN* contains a <local or schema qualifier>, *LSQ*, then *TTD* shall be contained in a <module> *M* and *LSQ* shall be either "MODULE" or the <module name> of *M*.

   b) If *ADTN* does not contain a <local or schema qualifier>, then *TTD* shall be contained in a <module> *M* and "MODULE" is implicit.

2) The descriptor of the abstract data type defined by a <temporary abstract data type declaration> includes the <abstract data type name>, and the other components as specified for <abstract data type body>.

Access Rules

None.

General Rules

1) If <temporary abstract data type declaration> is contained in a <module> *M*, then let *U* be the implementation-dependent <schema name> that is effectively derived from the implementation-dependent SQL-session identifier associated with the SQL-session and an implementation-dependent name associated with *M*.

2) The declaration of *T* within a module is effectively equivalent to the definition of a persistent abstract data type *U.QI*, where *QI* is the <qualified identifier> of <abstract data type name>. Within the module, any reference to MODULE.*T* is equivalent to a reference to *U.T*.

3) The definition of a <temporary abstract data type> persists for the duration of the SQL-session. The termination of the SQL-session is effectively followed by the execution of the following <drop data type statement> with the current authorization identifier and current <schema name> *U* without further Access Rule checking:

   DROP DATA TYPE *QI* CASCADE

4) Let *A* be the <authorization identifier> of the current SQL-session.

5) A set of privilege descriptors is created that defines the USAGE privilege for, and the UNDER privilege on, the temporary abstract data type. These privileges are not grantable. The grantor for these privilege descriptors is the special grantor value "_SYSTEM".

6) The definition of a <temporary abstract data type> does not appear in any view of the Information Schema.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <temporary abstract data type declaration>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

13.13 <temporary view declaration>

Function

Define a temporary viewed table whose definition exists for the duration of the current SQL-session.

Format

```
<temporary view declaration> ::=
       DECLARE TEMPORARY VIEW <table name>
           [ <left paren> <view column list> <right paren> ]
           AS <query expression>
```

Syntax Rules

1) Let *VN* be the <table name> of a <temporary view declaration> *TVD* and let *V* be the <qualified identifier> of *VN*.

   Case:

   a) If *VN* contains a <local or schema qualifier> *LSQ*, then *TVD* shall be contained in a <module> *M* and *LSQ* shall be "MODULE".

   b) If *VN* does not contain a <local or schema qualifier>, then *TVD* shall be contained in a <module> *M* and "MODULE" is implicit.

2) If a <temporary view declaration> is contained in a <module> *M*, then

   a) If *VN* contains a <local or schema qualifier>, then it shall be "MODULE".

   b) The <qualified identifier> of *VN* shall be different from the <qualified identifier> of the <table name> of any other <temporary table declaration> or <temporary view declaration> that is contained in *M*.

3) If the <query expression> is inherently updatable, then the temporary viewed table is an inherently updatable table. Otherwise, it is not an inherently updatable table.

4) If any two columns in the table specified by the <query expression> have the same <column name>, or if any column of that table has an implementation-dependent name, then a <view column list> shall be specified.

5) The same <column name> shall not be specified more than once in the <view column list>.

6) The number of <column name>s in the <view column list> shall be the same as the degree of the table specified by the <query expression>.

Access Rules

None.

General Rules

1) Let *U* be the implementation-dependent <schema name> that is effectively derived from the implementation-dependent SQL-session identifier associated with the SQL-session and an implementation-dependent name associated with the <module> that contains the <temporary table declaration>.

2) The specification of *V* in a module is effectively equivalent to the definition of a view *U.V*. Within the module, any reference to MODULE.*V* is equivalent to a reference to *U.V*.

3) A <temporary view declaration> defines a temporary viewed table. The temporary viewed table *TV* is the table that would result if the <query expression> were evaluated. Whether a temporary viewed table is materialized is implementation-dependent.

4) If the definition of a temporary view has not been dropped before the end of the SQL-session, then the definition of a temporary view persists for the duration of the SQL-session. The termination of the SQL-session is effectively followed by an <SQL schema statement> in a <module> with <module authorization identifier> *U* that specifies DROP VIEW *V*.

5) A view descriptor is created that describes *TV*. The view descriptor includes the <table name>, the <query expression>, and column descriptors taken from the tables specified by the <query expression>. If a <view column list> is specified, then the <column name> of the *i*-th column of the view is the *i*-th <column name> in that <view column list>. Otherwise, the <column name>s of the view are the <column name>s of the table specified by the <query expression>.

6) Let *A* be the <authorization identifier> of the current SQL-session.

7) A set of privilege descriptors is created that defines the SELECT and REFERENCES privileges on *TV* to *A* and SELECT and REFERENCES privileges for each column of *TV* to *A*. These privileges are not grantable. The grantor for each of these privileges is set to the special grantor value "_SYSTEM".

8) If *TV* is inherently updatable, then let *UT* be the underlying table of the <query expression>.

9) If *TV* is inherently updatable, then:

   a) A set of privilege descriptors is created that defines the privileges INSERT, UPDATE, and DELETE on *TV* that are the applicable privileges on *UT* to *A*. These privileges are not grantable. The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM".

   b) For every column in *TV*:

      i) There is a corresponding column in *UT* from which the column of *TV* is derived. Let *CV* and *CT* be the column names of *TV* and *T* respectively.

      ii) A set of privilege descriptors is created that defines the privileges INSERT(*CV*) and UPDATE(*CV*) on *TV* that correspond to the privileges INSERT(*CT*) and UPDATE(*CT*) on *UT*. These privileges are not grantable. The grantor for each of these privilege descriptors is set to the special grantor value "_SYSTEM".

   c) If *TV* is inherently updatable, then for each row in *TV* there is a corresponding row in *UT* from which the row of *TV* is derived and for each column in *TV* there is a corresponding column in *UT* from which the column of *TV* is derived. The insertion of a row into *TV* is an insertion of a corresponding row into *UT*. The deletion of a row from *TV* is a deletion of the

corresponding row in *UT*. The updating of a column of a row in *TV* is an updating of the corresponding column of the corresponding row in *UT*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <temporary view declaration>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

13.14 <free locator statement>

Function

Remove the association between a locator variable and the value that is represented by that locator.

Format

```
<free locator statement> ::=
      FREE LOCATOR <embedded variable name> [ { <comma> <embedded variablenam > }... ]
```

Syntax Rules

1) The data type of <embedded variable name> shall be character large object locator or binary large object locator.

2) Let *EV* be the embedded variable identified by <embedded variable name>.

3) If the value of *EV* does not specify a locator established within the current SQL-transaction, then an exception condition is raised: *locator exception—invalid specification*.

Access Rules

None.

General Rules

1) The locator identified by the value of <embedded variable name> is destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <free locator statement>

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14 Transaction management

14.1 <start transaction statement>

Function

Start an SQL-transaction and set its attributes.

Format

```
<start transaction statement> ::=
       START TRANSACTION <transaction mode> [ { <comma> <transaction mode> }...]

<transaction mode> ::=
         <isolation level>
       | <transaction access mode>
       | <diagnostics size>

<transaction access mode> ::=
         READ ONLY
       | READ WRITE

<isolation level> ::=
       ISOLATION LEVEL <level of isolation>

<level of isolation> ::=
         READ UNCOMMITTED
       | READ COMMITTED
       | REPEATABLE READ
       | SERIALIZABLE

<diagnostics size> ::=
       DIAGNOSTICS SIZE <number of conditions>

<number of conditions> ::= <simple value specification>
```

Syntax Rules

1) No <transaction mode> shall be specified more than once.

2) If an <isolation level> is not specified, then a <level of isolation> of ISOLATION LEVEL SERIALIZABLE is implicit.

3) If READ WRITE is specified, then the <level of isolation> shall not be READ UNCOMMITTED.

4) If a <transaction access mode> is not specified and a <level of isolation> of READ UNCOMMITTED is specified, then READ ONLY is implicit. Otherwise, READ WRITE is implicit.

5) The data type of <number of conditions> shall be exact numeric with scale 0.

Access Rules

None.

General Rules

1) If a <start transaction statement> statement is executed when an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state—active SQL-transaction*.

2) If <number of conditions> is specified and is less than 1, then an exception condition is raised: *invalid condition number*.

3) Let *TXN* be the SQL-transaction that is started by the <start transaction statement>.

   Note: The characteristics of a transaction begun by a <start transaction statement> are as specified in these General Rules regardless of the characteristics specified by any preceding <set transaction statement>. That is, even if one or more characteristics are omitted by the <start transaction statement>, the defaults specified in the Syntax Rules of this Subclause are effective and are not affected by any (preceding) <set transaction statement>.

4) If READ ONLY is specified, then the access mode of *TXN* is set to *read-only*. If READ WRITE is specified, then the access mode of *TXN* is set to *read-write*.

5) The isolation level of *TXN* is set to an implementation-defined isolation level that will not exhibit any of the phenomena that the explicit or implicit <level of isolation> would not exhibit, as specified in Table 10, "SQL-transaction isolation levels and the three phenomena".

6) If <number of conditions> is specified, then the diagnostics area limit of *TXN* is set to <number of conditions>.

7) If <number of conditions> is not specified, then the diagnostics area limit of *TXN* is set to an implementation-dependent value not less than 1.

8) *TXN* is started.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not contain any <start transaction statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.2 <set transaction statement>

Function

Set the attributes of the next SQL-transaction for the SQL-agent.

Note: This statement has no effect on any SQL-transactions subsequent to the next SQL-transaction.

Format

```
<set transaction statement> ::=
      SET [ LOCAL ] <transaction attributes>

<transaction attributes> ::=
      TRANSACTION <transaction mode> [ { <comma> <transaction mode> }... ]
```

Syntax Rules

1) If LOCAL is specified, then <number of conditions> shall not be specified.

Access Rules

None.

General Rules

1) Case:

   a) If a <set transaction statement> that does not specify LOCAL is executed, then

      Case:

      i) If an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state—active SQL-transaction*.

      ii) If an SQL-transaction is not currently active, then if there are any holdable-cursors remaining open from the previous SQL-transaction and the isolation level of the previous SQL-transaction is not the same as the isolation level determined by the <level of isolation>, then an exception condition is raised: *invalid transaction state— held cursor requires same isolation level*.

   b) If a <set transaction statement> that specifies LOCAL is executed, then:

      i) If the implementation does not support SQL-transactions that affect more than one SQL-server, then an exception condition is raised: *feature not supported—multiple server transactions*.

      ii) If there is no SQL-transaction that is currently active, then an exception condition is raised: *invalid transaction state— no active SQL-transaction for branch transaction*.

      iii) If there is an active SQL-transaction and there has been a transaction-initiating SQL-statement executed at the current SQL-connection in the context of the active SQL-transaction, then an exception condition is raised: *invalid transaction state—branch transaction already active*.

iv) If the transaction access mode of the SQL-transaction is read-only and <transaction access mode> specifies READ WRITE, then an exception condition is raised: *invalid transaction state— inappropriate access mode for branch transaction*.

v) If the isolation level of the SQL-transaction is SERIALIZABLE and <level of isolation> specifies anything except SERIALIZABLE, then an exception condition is raised: *invalid transaction state— inappropriate isolation level for branch transaction*.

vi) If the isolation level of the SQL-transaction is REPEATABLE READ and <level of isolation> specifies anything except REPEATABLE READ or SERIALIZABLE, then an exception condition is raised: *invalid transaction state— inappropriate isolation level for branch transaction*.

vii) If the isolation level of the SQL-transaction is READ COMMITTED and <level of isolation> specifies READ UNCOMMITTED, then an exception condition is raised: *invalid transaction state— inappropriate isolation level for branch transaction*.

Note: If the isolation level of the SQL-transaction is READ UNCOMMITTED, then any <level of isolation> is permissable.

2) If <number of conditions> is specified and is less than 1, then an exception condition is raised: *invalid condition number*.

3) Case:

a) If LOCAL is not specified, then let *TXN* be the next SQL-transaction for the SQL-agent.

b) Otherwise, let *TXN* be the branch of the active SQL-transaction at the current SQL-connection.

4) If READ ONLY is specified, then the access mode of *TXN* is set to *read-only*. If READ WRITE is specified, then the access mode of *TXN* is set to *read-write*.

5) The isolation level of *TXN* is set to an implementation-defined isolation level that will not exhibit any of the phenomena that the explicit or implicit <level of isolation> would not exhibit, as specified in Table 10, "SQL-transaction isolation levels and the three phenomena".

6) If <number of conditions> is specified, then the diagnostics area limit of *TXN* is set to <number of conditions>.

7) If <number of conditions> is not specified, then the diagnostics area limit of *TXN* is set to an implementation-dependent value not less than 1.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) Conforming Full SQL language shall not specify LOCAL.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any <set transaction statement>.

14.3 <set constraints mode statement>

Function

If an SQL-transaction is currently active, then set the constraint mode for that SQL-transaction in the current SQL-session. If no SQL-transaction is currently active, then set the constraint mode for the next SQL-transaction in the current SQL-session for the SQL-agent.

Note: This statement has no effect on any SQL-transactions subsequent to this SQL-transaction.

Format

```
<set constraints mode statement> ::=
       SET CONSTRAINTS <constraint name list> { DEFERRED | IMMEDIATE }

<constraint name list> ::=
         ALL
       | <constraint name> [ { <comma> <constraint name> }... ]
```

Syntax Rules

1) If a <constraint name> is specified, then it shall identify a constraint.

2) The constraint identified by <constraint name> shall be DEFERRABLE.

Access Rules

None.

General Rules

1) If an SQL-transaction is currently active, then let *TXN* be the currently active SQL-transaction. Otherwise, let *TXN* be the next SQL-transaction for the SQL-agent.

2) If IMMEDIATE is specified, and if the cursor mode of *TXN* is *cascade off*, then an exception condition is raised: *invalid constraint mode state*.

3) If IMMEDIATE is specified, then

   Case:

   a) If ALL is specified, then the constraint mode in *TXN* of all constraints that are DEFERRABLE is set to *immediate*.

   b) Otherwise, the constraint mode in *TXN* for the constraints identified by the <constraint name>s in the <constraint name list> is set to *immediate*.

4) If DEFERRED is specified, then

   Case:

   a) If ALL is specified, then the constraint mode in *TXN* of all constraints that are DEFERRABLE is set to *deferred*.

b) Otherwise, the constraint mode in *TXN* for the constraints identified by the <constraint name>s in the <constraint name list> is set to *deferred*.

Leveling Rules

1) The following restrictions apply for Full SQL:

None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

a) Conforming Intermediate SQL language shall not contain any <set constraints mode statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

14.4 <test completion statement>

Function

Test the completion of one or more asynchronous SQL-statements.

Format

```
<test completion statement> ::=
      { TEST | WAIT } { ALL | ANY | <async statement identifier list> } COMPLETION

<async statement identifier list> ::=
      <left paren> <async statement identifier>
          [ { <comma> <async statement identifier> }... ] <right paren>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) If there is no outstanding asynchronous <SQL procedure statement>, then an exception condition is raised: *no outstanding asynchronous SQL statement*.

2) Case:

   a) If WAIT is specified, then

      Case:

      i) If ALL is specified, then the DBMS shall wait until all outstanding asynchronous <SQL procedure statement>s have completed. All such <SQL procedure statement>s are said to have been *successfully tested for completion*.

      ii) If ANY is specified, then the DBMS shall wait until at least one outstanding asynchronous <SQL procedure statement> has completed that has not already been successfully tested for completion. The <SQL procedure statement> that satisfies the WAIT ANY is said to have been *successfully tested for completion*.

      iii) Otherwise, the DBMS shall wait until all asynchronous <SQL procedure statement>s identified by <async statement identifier>s in the <async statement identifier list> have completed. All such <SQL procedure statement>s are said to have been *successfully tested for completion*.

   b) If TEST is specified, then

Case:

i) If ALL is specified and if any asynchronous <SQL procedure statement> is outstanding, then an exception condition is raised: *SQL statement not yet complete*. Any <SQL procedure statement>s that have completed are said to have been *successfully tested for completion*.

ii) If ANY is specified and if no asynchronous <SQL procedure statement> has completed without having been successfully tested for completion, then an exception condition is raised: *SQL statement not yet complete*. Any <SQL procedure statement>s that have completed are said to have been *successfully tested for completion*.

iii) Otherwise, if any of the asynchronous <SQL procedure statement>s identified by <async statement identifier>s in the <async statement identifier list> have not completed, then an exception condition is raised: *SQL statement not yet complete*. Any <SQL procedure statement>s identified by <async statement identifier>s in the <async statement identifier list> that have completed are said to have been *successfully tested for completion*.

3) If any <SQL procedure statement> successfully tested for completion by the <test completion statement> received an SQLSTATE value other than "00000", then the SQLSTATE subcode is set to *asynchronous SQL statement returned an SQLSTATE value*.

4) For every <routine> that contained an asynchronous <SQL procedure statement> successfully tested for completion by the <test completion statement>, all the parameters, including status parameters, are set to the values that they would have received had the <SQL procedure statement> been executed synchronously.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <test completion statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.5 <savepoint statement>

Function

Establish a savepoint.

Format

```
<savepoint statement> ::= SAVEPOINT <savepoint specifier>
```

Syntax Rules

1) If the <savepoint specifier> is specified as a <simple target specification>, let *T* be the <simple target specification>. The data type of *T* shall be exact numeric with scale 0.

Access Rules

None.

General Rules

1) If <savepoint specifier> is specified as <savepoint name>, then let *S* be the <identifier> specified as <savepoint name>; otherwise, let *S* be the value of *T*.

2) If <savepoint specifier> is specified as <simple target specification> and *S* is not 0 and does not identify an existing savepoint established within the current SQL-transaction, then an exception condition is raised: *savepoint exception—invalid specification*.

3) If *S* identifies an existing savepoint established within the current SQL-transaction, then that savepoint is destroyed.

4) If the number of savepoints that now exist within the current SQL-transaction is equal to the implementation-defined maximum number of savepoints per SQL-transaction, then an exception condition is raised: *savepoint exception—too many*.

5) If <savepoint specifier> is specified as <simple target specification>, then *S* is set to an implementation-dependent value that is non-0 and different from all other values that have been used to identify savepoints within the current SQL-transaction.

6) A savepoint is established at the current point in the current SQL-transaction and *S* is assigned as the identifier of that savepoint.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <savepoint statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

14.6 <release savepoint statement>

Function

Destroy a savepoint.

Format

```
<release savepoint statement> ::=
      RELEASE SAVEPOINT <savepoint specifier>
```

Syntax Rules

1) If <savepoint specifier> is specified as <simple target specification>, then let *T* be that <simple target specification>. The data type of *T* shall be exact numeric.

Access Rules

None.

General Rules

1) If <savepoint specifier> is specified as <savepoint name>, then let *S* be the identifier specified as <savepoint name>; otherwise, let *S* be the value of *T*.

2) If *S* does not identify a savepoint defined within the current SQL-transaction, then an exception condition is raised: *savepoint exception—invalid specification*.

3) The savepoint identified by *S* and all savepoints established in the current SQL-transaction subsequent to the establishment of S are destroyed.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <release savepoint statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.7 <commit statement>

Function

Terminate the current SQL-transaction with commit.

Format

```
<commit statement> ::=
       COMMIT [ WORK ] [ AND [ NO ] CHAIN ]
```

Syntax Rules

1) If neither AND CHAIN nor AND NO CHAIN is specified, then AND NO CHAIN is implicit.

Access Rules

None.

General Rules

1) If the current SQL-transaction is part of an encompassing transaction that is controlled by an agent other than the SQL-agent, then an exception condition is raised: *invalid transaction termination*.

2) If an atomic execution context is active, then an exception condition is raised: *invalid transaction termination*.

3) The following <test completion statement> is effectively executed:

   WAIT ALL COMPLETION

4) For every open cursor that is not a holdable-cursor *CR* in any <module> associated with the current SQL-transaction, the following statement is implicitly executed:

   CLOSE *CR*

5) For every temporary table in any <module> associated with the current SQL-transaction that specifies the ON COMMIT DELETE option and that was updated by the current SQL-transaction, the execution of the <commit statement> is effectively preceded by the execuion of a <delete statement: searched> that specifies DELETE FROM *T*, where *T* is the <table name> of that temporary table. If the cursor mode of the current SQL-transaction is *cascade off*, then the cursor mode of the current SQL-transaction is set to *cascade on*.

6) The effects specified in the General Rules of Subclause 14.3, "<set constraints mode statement>" occur as if the statement SET CONSTRAINTS ALL IMMEDIATE were executed for each active SQL-connection.

7) Any triggered actions that were deferred are effectively executed.

8) Case:

   a) If any constraint is not satisfied, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback—integrity constraint violation*.

   b) If the execution of any <triggered SQL statement> is unsuccessful, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback—triggered action exception*.

   c) If any other error preventing commitment of the SQL-transaction has occurred, then any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled and an exception condition is raised: *transaction rollback* with an implementation-defined subclass value.

   d) Otherwise, any changes to SQL-data or schemas that were made by the current SQL-transaction are made accessible to all concurrent and subsequent SQL-transactions.

9) Any savepoints established in the current SQL-transaction are destroyed.

10) Any locators generated in the current SQL-transaction are destroyed.

11) The current SQL-transaction is terminated. If AND CHAIN was specified, then a new SQL-transaction is initiated with the same access mode, isolation level, and diagnostics area size as the SQL-transaction just terminated. Any branch transactions of the SQL-transaction are initiated with the same access mode, isolation level, and diagnostics area size as the corresponding branch of the SQL-transaction just terminated.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not specify CHAIN.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) In conforming Entry SQL language, WORK shall be specified.

14.8 <rollback statement>

Function

Terminate the current SQL-transaction with rollback, or rollback all database actions since the establishment of a savepoint.

Format

```
<rollback statement> ::=
       ROLLBACK [ WORK ] [ AND[ NO ]  CHAIN ]
         [ <savepoint clause> ]

<savepoint clause> ::=
       TO SAVEPOINT <savepoint specifier>

<savepoint specifier> ::=
         <savepoint name>
       | <simple target specification>
```

Syntax Rules

1) If <savepoint specifier> is specified as <simple target specification>, then let *T* be that <simple target specification>. The data type of *T* shall be exact numeric.

2) If AND CHAIN is specified, then <savepoint clause> shall not be specified.

3) If neither AND CHAIN nor AND NO CHAIN is specified, then AND NO CHAIN is implicit.

Access Rules

None.

General Rules

1) If the current SQL-transaction is part of an encompassing transaction that is controlled by an agent other than the SQL-agent and the <rollback statement> is not being implicitly executed, then an exception condition is raised: *invalid transaction termination*.

2) Any outstanding asynchronous <SQL procedure statement> is terminated.

3) If a <savepoint clause> is not specified, then:

   a) If an atomic execution context is active, then an exception condition is raised: *invalid transaction termination*.

   b) Any outstanding asynchronous <SQL procedure statement> is terminated.

   c) Any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled.

   d) All savepoints defined by the current SQL-transaction are destroyed.

   e) Any locators generated in the current SQL-transaction are destroyed.

f) Any open cursors in any <module> associated with the current SQL-transaction that were opened by the current SQL-transaction subsequent to the establishment of *S* are closed.

g) The current SQL-transaction is terminated. If AND CHAIN was specified, then a new SQL-transaction is initiated with the same access mode, isolation level, and diagnostics area size as the SQL-transaction just terminated. Any branch transactions of the SQL-transaction are initiated with the same access mode, isolation level, and diagnostics area size as the corresponding branch of the SQL-transaction just terminated.

4) If a <savepoint clause> is specified, then:

a) If <savepoint name> is specified as <simple target specification>, then let *S* be the <identifier> specified as <savepoint name>; otherwise, let *S* be the value of *T*.

b) If *S* does not specify a savepoint established within the current SQL-transaction, then an exception condition is raised: *savepoint exception—invalid specification*.

c) If an atomic execution context is active when *S* is executed, and *S* specifies a savepoint established before the beginning of the most recent atomic execution context, then an exception condition is raised: *savepoint exception—invalid specification*.

d) Any changes to SQL-data or schemas that were made by the current SQL-transaction subsequent to the establishment of *S* are canceled.

e) All savepoints established by the current SQL-transaction subsequent to the establishment of *S* are destroyed.

f) For every open cursor *CR* in any <module> associated with the current SQL-transaction that was opened subsequent to the establishment of *S*, the following statement is implicitly executed:

CLOSE *CR*

****Editor's Note****

The wording of the preceding rule in light of SQL3 savepoints has been identified as a Possible Problem. See Possible Problem **164** in the Editor's Notes. Furthermore, this rule seems redundant with a rule just above!

g) The status of any open cursors in any <module> associated with the current SQL-transaction that were opened by the current SQL-transaction before the establishment of *S* is implementation-defined.

Note: The current SQL-transaction is not terminated, and there is no other effect on the SQL-data or schemas.

Leveling Rules

1) The following restrictions apply for Full SQL:

a) A <rollback statement> shall contain no <savepoint clause>.

b) Conforming Full SQL language shall not specify CHAIN.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) In conforming Entry SQL language, WORK shall be specified.

15 Connection management

15.1 <connect statement>

Function

Establish an SQL-session.

Format

```
<connect statement> ::=
       CONNECT TO <connection target>

<connection target> ::=
         <SQL-server name>
           [ AS <connection name> ]
           [ USER <user name> ]
       | DEFAULT
```

Syntax Rules

1) If <user name> is not specified, then an implementation-defined <user name> for the SQL-connection is implicit.

Access Rules

None.

General Rules

1) If a <connect statement> is executed after the first transaction-initiating SQL-statement executed by the current SQL-transaction and the implementation does not support transactions that affect more than one SQL-server, then an exception condition is raised: *feature not supported—multiple server transactions*.

2) If <user name> is specified, then let *S* be the character string that is the value of <user name> and let *V* be the character string that is the value of

   TRIM (BOTH ' ' FROM *S*)

3) If *V* does not conform to the Format and Syntax Rules of an <authorization identifier>, then an exception condition is raised: *invalid authorization specification*.

4) If the <module> that contains the <routine> that contains the <connect statement> specifies a <module authorization identifier>, then whether or not <user name> must be identical to that <module authorization identifier> is implementation-defined, as are any other restrictions on the value of <user name>. Otherwise, any restrictions on the value of <user name> are implementation-defined.

5) If the value of <user name> violates the implementation-defined restrictions, then an exception condition is raised: *invalid authorization specification*.

6) If <connection name> was specified, then let *CV* be the value of the <simple value specification> immediately contained in <connection name>. If neither DEFAULT nor <connection name> were specified, then let *CV* be the value of <SQL-server name>. Let *CN* be the result of

   TRIM (BOTH ' ' FROM *CV*)

   If *CN* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid connection name*.

7) If an SQL-connection with name *CN* has already been established by the current SQL-agent and has not been disconnected, or if DEFAULT is specified and a default SQL-connection has already been established by the current SQL-agent and has not been disconnected, then an exception condition is raised: *connection exception—connection name in use*.

8) Case:

   a) If DEFAULT is specified, then the default SQL-session is initiated and associated with the default SQL-server. The method by which the default SQL-server is determined is implementation-defined.

   b) Otherwise, an SQL-session is initiated and associated with the SQL-server identified by <SQL-server name>. The method by which <SQL-server name> is used to determine the appropriate SQL-server is implementation-defined.

9) If the <connect statement> successfully initiates an SQL-session, then:

   a) The current SQL-connection and current SQL-session, if any, become a dormant SQL-connection and a dormant SQL-session, respectively. The SQL-session context information is preserved and is not affected in any way by operations performed over the initiated SQL-connection.

      Note: The SQL-session context information is defined in Subclause 4.41, "SQL-sessions".

   b) The SQL-session initiated by the <connect statement> becomes the current SQL-session and the SQL-connection established to that SQL-session becomes the current SQL-connection.

   Note: If the <connect statement> fails to initiate an SQL-session, then the current SQL-connection and current SQL-session, if any, remain unchanged.

10) If the SQL-client cannot establish the SQL-connection, then an exception condition is raised: *connection exception— SQL-client unable to establish SQL-connection*.

11) If the SQL-server rejects the establishment of the SQL-connection, then an exception condition is raised: *connection exception— SQL-server rejected establishment of SQL-connection*.

12) The SQL-server for the subsequent execution of <routine>s in any <module>s associated with the SQL-agent is set to the SQL-server identified by <SQL-server name>.

13) The SQL-session <authorization identifier> is set to <user name>.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <connect statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

15.2 <set connection statement>

Function

Select an SQL-connection from the available SQL-connections.

Format

```
<set connection statement> ::=
       SET CONNECTION <connection object>

<connection object> ::=
         DEFAULT
       | <connection name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) If a <set connection statement> is executed after the first transaction-initiating SQL-statement executed by the current SQL-transaction and the implementation does not support transactions that affect more than one SQL-server, then an exception condition is raised: *feature not supported—multiple server transactions*.

2) Case:

   a) If DEFAULT is specified and there is no default SQL-connection that is current or dormant for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

   b) Otherwise, if <connection name> does not identify an SQL-session that is current or dormant for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

3) If the SQL-connection identified by <connection object> cannot be selected, then an exception condition is raised: *connection exception—connection failure*.

4) The current SQL-connection and current SQL-session become a dormant SQL-connection and a dormant SQL-session, respectively. The SQL-session context information is preserved and is not affected in any way by operations performed over the selected SQL-connection.

   Note: The SQL-session context information is defined in Subclause 4.41, "SQL-sessions".

5) The SQL-connection identified by <connection object> becomes the current SQL-connection and the SQL-session associated with that SQL-connection becomes the currenct SQL-session. All SQL-session context information is restored to the same state as at the time the SQL-connection became dormant.

Note: The SQL-session context information is defined in Subclause 4.41, "SQL-sessions".

6) The SQL-server for the subsequent execution of <routine>s in any <module>s associated with the SQL-agent are set to that of the current SQL-connection.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <set connection statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

15.3 <disconnect statement>

Function

Terminate an SQL-connection.

Format

```
<disconnect statement> ::=
       DISCONNECT <disconnect object>

<disconnect object> ::=
         <connection object>
       | ALL
       | CURRENT
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) If <connection name> is specified and <connection name> does not identify an SQL-connection that is current or dormant for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

2) If DEFAULT is specified and there is no default SQL-connection that is current or dormant for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

3) If CURRENT is specified and there is no current SQL-connection for the current SQL-agent, then an exception condition is raised: *connection exception—connection does not exist*.

4) Let *C* be the current SQL-connection.

5) Let *L* be a list of SQL-connections. If a <connection name> is specified, then *L* is that SQL-connection. If CURRENT is specified, then *L* is the current SQL-connection. If ALL is specified, then *L* is a list representing every SQL-connection that is current or dormant for the current SQL-agent, in an implementation-dependent order. If DEFAULT is specified, then *L* is the default SQL-connection.

6) If any SQL-connection in *L* is active, then an exception condition is raised: *invalid transaction state—active SQL-transaction*.

7) For every SQL-connection *C1* in *L*, treating the SQL-session *S1* identified by *C1* as the current SQL-session, all of the actions that are required after the last call of a <routine> by an SQL-agent, except for the execution of a <rollback statement> or a <commit statement>, are performed. *C1* is terminated, regardless of any exception condition that might occur during the disconnection process.

Note: See the General Rules of Subclause 12.2, "<module>", for the actions to be performed after the last call of a <routine> by an SQL-agent.

8) If any error is detected during execution of a <disconnect statement>, then a completion condition is raised: *warning—disconnect error*.

9) If *C* is contained in *L*, then there is no current SQL-connection following the execution of the <disconnect statement>. Otherwise, *C* remains the current SQL-connection.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not contain any <disconnect statement>.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

16 Session management

16.1 <set session characteristics statement>

Function

Set one or more characteristics for the current SQL-session.

Format

```
<set session characteristics statement> ::=
      SET SESSION CHARACTERISTICS AS
        <session attribute list>

<session attribute list> ::=
      <session attribute> [ { <comma> <session attribute> }... ]

<session attribute> ::=
        <transaction attributes>
```

Syntax Rules

1) No <session attribute> shall be specified more than once in a <session attribute list>.

Access Rules

None.

General Rules

1) If <transaction attributes> is specified in a <session attribute list>, then the enduring transaction characteristics of the SQL-session are set to the values explicitly specified in the <transaction attributes>; enduring characteristics corresponding to <transaction attributes> values not explicitly specified are unchanged.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL Language shall contain no <set session characteristics statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions.

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions.

   None.

16.2 <set session authorization identifier statement>

Function

Set the <authorization identifier> of the current SQL-session.

Format

```
<set session authorization identifier statement> ::=
       SET SESSION AUTHORIZATION <value specification>
```

Syntax Rules

1) The <data type> of the <value specification> shall be an SQL character data type.

Access Rules

None.

General Rules

1) If a <set session authorization identifier statement> is executed and an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state—active SQL-transaction*.

2) Let *S* be the character string that is the value of the <value specification> and let *V* be the character string that is the value of

   TRIM (BOTH ' ' FROM *S*)

3) If *V* does not conform to the Format and Syntax Rules of an <authorization identifier>, then an exception condition is raised: *invalid authorization specification*.

4) Whether or not the <authorization identifier> for the SQL-session can be set to an <authorization identifier> other than the <authorization identifier> of the SQL-session when the SQL-session is started is implementation-defined, as are any restrictions pertaining to such changes.

5) If the current <authorization identifier> is restricted from setting the <authorization identifier> to the specified value, then an exception condition is raised: *invalid authorization specification*.

6) Let *T* be any temporary table or temporary view declared in any <module> invoked during the currently active SQL-session. In all the privilege descriptors for *T* and for each of the columns of *T*, the <authorization identifier> is set to *V*.

7) The <authorization identifier> of the current SQL-session is set to *V*.

ANSI Only–SQL3

8) If there exists a default role descriptor that identifies the <authorization identifier> of the <set session authorization identifier statement> as a grantee, then the role identified by the default role of the default role descriptor is enabled.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <set session authorization identifier statement>.

16.3 <set role statement>

Function

Set enabled applicable roles for the current SQL-session.

Format

```
<set role statement> ::=
      SET ROLE <role specification>

<role specification> ::=
```

ISO Only--SQL3

```
  <value specification>
```

ANSI Only---caused by ISO changes not yet considered by ANSI

```
  <role name>

| NONE
```

ANSI Only--SQL3

```
| DEFAULT
```

Syntax Rules

ISO Only–SQL3

1) The <data type> of the <value specification> shall be an SQL character data type.

ANSI Only—caused by ISO changes not yet considered by ANSI

None.

Access Rules

ISO Only–SQL3

None.

ANSI Only—caused by ISO changes not yet considered by ANSI

1) A role authorization descriptor shall exist that indicates that the role identified by <role name> has been granted to the <authorization identifier> of the current SQL-session or to PUBLIC.

2) If DEFAULT is specified, then a default role descriptor shall exist that indicate the current <authorization identifier> as a grantee.

General Rules

1) If a <set role statement> is executed and an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state—active SQL-transaction*.

ISO Only–SQL3

2) Let *S* be the character striang that is the value of the <value specification> and let *V* be the character string that is the value of

   TRIM (BOTH ' ' FROM *S*)

3) If *V* does not conform to the Format and Syntax Rules of a <role name>, then an excewption condition is raised: *invalid role specification*.

4) If no role authorization descriptor exists that indicates that the role identified by *V* has been granted to either the <authorization identifier> of the current SQL-session or to PUBLIC, then an exception condition is raised: *invalid role specification>*.

5) If NONE is specified, then let *R* be the empty set. Otherwise, let *R* be the set of roles consisting of the role identified by *V* and all other roles contained in that role.

ANSI Only—caused by ISO changes not yet considered by ANSI

6) If <role name> is specified, then let *R* be the set of roles consisting of the role identified by <role name> and all other roles contained in that role. Otherwise, let *R* be the empty set.

7) If DEFAULT is specified then let *R* be the <default role> for the current SQL-session <authorization identifier>.

8) The enabled applicable roles are set to *R*.

9) The set of applicable privileges for the <authorization identifier> of the current SQL-session is set to contain those privileges defined by privilege descriptors associated with the roles in *R* or defined by privilege descriptors associated with that <authorization identifier>, together with those defined by privilege descriptors associated with PUBLIC.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <set role statement>.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

16.4 <set local time zone statement>

Function

Set the default local time zone displacement for the current SQL-session.

Format

```
<set local time zone statement> ::=
      SET TIME ZONE <set time zone value>

<set time zone value> ::=
        <interval value expression>
      | LOCAL
```

Syntax Rules

1) The <data type> of the <interval value expression> immediately contained in the <set time zone value> shall be INTERVAL HOUR TO MINUTE.

Access Rules

None.

General Rules

1) Case:

   a) If LOCAL is specified, then the default local time zone displacement of the current SQL-session is set to the original implementation-defined default local time zone displacement that was established when the current SQL-session was started.

   b) Otherwise,

      Case:

      i) If the value of the <interval value expression> is not the null value and is between INTERVAL –'12:59' and INTERVAL +'13:00', then the default local time zone displacement of the current SQL-session is set to the value of the <interval value expression>.

      ii) Otherwise, an exception condition is raised: *data exception—invalid time zone displacement value*.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restsrictions:

   a) Conforming Entry SQL language shall not contain any <set local time zone statement>.

17 Diagnostics management

17.1 <get diagnostics statement>

Function

Get exception or completion condition information from the diagnostics area.

Format

```
<get diagnostics statement> ::=
      GET DIAGNOSTICS <sql diagnostics information>

<sql diagnostics information> ::=
        <statement information>
      | <condition information>

<statement information> ::=
      <statement information item> [ { <comma> <statement information item> }... ]

<statement information item> ::=
      <simple target specification> <equals operator> <statement information item name>


<statement information item name> ::=
        NUMBER
      | MORE
      | COMMAND_FUNCTION
      | COMMAND_FUNCTION_CODE
      | ROW_COUNT
      | TRANSACTIONS_COMMITTED
      | TRANSACTIONS_ROLLED_BACK
      | TRANSACTION_ACTIVE

<condition information> ::=
      EXCEPTION <condition number>
        <condition information item> [ { <comma> <condition information item> }... ]

<condition information item> ::=
      <simple target specification> <equals operator> <condition information item name>


<condition information item name> ::=
        CONDITION_NUMBER
      | RETURNED_SQLSTATE
      | CLASS_ORIGIN
      | SUBCLASS_ORIGIN
      | SERVER_NAME
      | CONNECTION_NAME
      | CONSTRAINT_CATALOG
      | CONSTRAINT_SCHEMA
      | CONSTRAINT_NAME
      | TRIGGER_CATALOG
      | TRIGGER_SCHEMA
      | TRIGGER_NAME
```

```
        | CATALOG_NAME
        | SCHEMA_NAME
        | TABLE_NAME
        | COLUMN_NAME
        | CURSOR_NAME
        | ROUTINE_CATALOG
        | ROUTINE_SCHEMA
        | ROUTINE_NAME
        | SPECIFIC_NAME
        | MESSAGE_TEXT
        | MESSAGE_LENGTH
        | MESSAGE_OCTET_LENGTH

<condition number> ::= <simple value specification>
```

Syntax Rules

1) The data type of a <simple target specification> contained in a <statement information item> or <condition information item> shall be the data type specified in Table 18, "<identifier>s for use with <get diagnostics statement>", for the corresponding <statement information item name> or <condition information item name>.

2) The data type of <condition number> shall be exact numeric with scale 0.

Table 18—<identifier>s for use with <get diagnostics statement>

<identifier>	Data Type
<statement information item name>s	
NUMBER	exact numeric with scale 0
MORE	character string (1)
COMMAND_FUNCTION	character varying (*L*)
COMMAND_FUNCTION_CODE	exact numeric with scale 0
ROW_COUNT	exact numeric with scale 0
TRANSACTIONS_COMMITTED	exact numeric with scale 0
TRANSACTIONS_ROLLED_BACK	exact numeric with scale 0
TRANSACTION_ACTIVE	exact numeric with scale 0
<condition information item name>s	
CONDITION_NUMBER	exact numeric with scale 0
RETURNED_SQLSTATE	character string (5)
CLASS_ORIGIN	character varying (*L*)
SUBCLASS_ORIGIN	character varying (*L*)
SERVER_NAME	character varying (*L*)
CONNECTION_NAME	character varying (*L*)
CONSTRAINT_CATALOG	character varying (*L*)
CONSTRAINT_SCHEMA	character varying (*L*)
CONSTRAINT_NAME	character varying (*L*)
TRIGGER_CATALOG	character varying (*L*)
TRIGGER_SCHEMA	character varying (*L*)
TRIGGER_NAME	character varying (*L*)
CATALOG_NAME	character varying (*L*)
SCHEMA_NAME	character varying (*L*)
TABLE_NAME	character varying (*L*)
COLUMN_NAME	character varying (*L*)
CURSOR_NAME	character varying (*L*)
ROUTINE_CATALOG	character varying (*L*)
ROUTINE_SCHEMA	character varying (*L*)
ROUTINE_NAME	character varying (*L*)
SPECIFIC_NAME	character varying (*L*)
PARAMETER_NAME	character varying (*L*)
MESSAGE_TEXT	character varying (*L*)
Where *L* is an implementation-defined integer not less than 128.	

(Continued on next page)

Table 18—<identifier>s for use with <get diagnostics statement> (Cont.)

<identifier>	Data Type
<condition information item name>s	
MESSAGE_LENGTH	exact numeric with scale 0
MESSAGE_OCTET_LENGTH	exact numeric with scale 0

Access Rules

None.

General Rules

1) Specification of <statement information item> retrieves information about the statement execution recorded in the diagnostics area into <simple target specification>.

   a) The value of NUMBER is the number of exception or completion conditions that have been stored in the diagnostics area as a result of executing the previous SQL-statement other than a <get diagnostics statement>.

      Note: The <get diagnostics statement> itself may return information via the SQLCODE or SQLSTATE parameters, but does not modify the previous contents of the diagnostics area.

   b) The value of MORE is:

      Y More conditions were raised during execution of the SQL-statement than have been stored in the diagnostics area.

      N All of the conditions that were raised during execution of the SQL-statement have been stored in the diagnostics area.

   c) The value of COMMAND_FUNCTION is the identification of the SQL-statement executed. Table 19, "SQL-statement character codes for use in the diagnostics area" specifies the identifier of the SQL-statements.

   d) The value of COMMAND_FUNCTION_CODE is a number identifying the SQL-statement executed. Table 19, "SQL-statement character codes for use in the diagnostics area" specifies the code for the SQL-statements. Positive values are reserved for SQL-statements defined by this
      ANSI American
      ISO International
      Standard; negative values are reserved for implementation-defined SQL-statements.

Table 19—SQL-statement character codes for use in the diagnostics area

SQL-statement	Identifier	Code
<abstract data type definition>	CREATE TYPE	83
<alter domain statement>	ALTER DOMAIN	3
<alter table statement>	ALTER TABLE	4
<assignment statement>	ASSIGNMENT	5

Table 19—SQL-statement character codes for use in the diagnostics area (Cont.)

SQL-statement	Identifier	Code
<assertion definition>	CREATE ASSERTION	6
<call statement>	CALL	7
<character set definition>	CREATE CHARACTER SET	8
<close statement>	CLOSE CURSOR	9
<collation definition>	CREATE COLLATION	10
<commit statement>	COMMIT WORK	11
<compound statement>	BEGIN END	12
<connect statement>	CONNECT	13
<create routine>	CREATE ROUTINE	14
• 1 table row deleted		
<delete statement: positioned>	DELETE CURSOR	18
<delete statement: searched>	DELETE WHERE	19
<disconnect statement>	DISCONNECT	22
<domain definition>	CREATE DOMAIN	23
<drop assertion statement>	DROP ASSERTION	24
<drop character set statement>	DROP CHARACTER SET	25
<drop collation statement>	DROP COLLATION	26
<drop domain statement>	DROP DOMAIN	27
<drop module statement>	DROP MODULE	28
<drop role statement>	DROP ROLE	29
<drop routine statement>	DROP ROUTINE	30
<drop schema statement>	DROP SCHEMA	31
<drop table statement>	DROP TABLE	32
<drop translation statement>	DROP TRANSLATION	33
<drop trigger statement>	DROP TRIGGER	34
<drop data type statement>	DROP DATA TYPE	35
<drop view statement>	DROP VIEW	36
<fetch statement>	FETCH	45
<for statement>	FOR	46
<grant statement>	GRANT	48
<grant role statement>	GRANT ROLE	49
<insert statement>	INSERT	50
• 1 table entry deleted.		
<open statement>	OPEN	53
<release savepoint statement>	RELEASE SAVEPOINT	57

Table 19—SQL-statement character codes for use in the diagnostics area (Cont.)

SQL-statement	Identifier	Code
<return statement>	RETURN	58
<revoke statement>	REVOKE	59
<revoke role statement>	REVOKE ROLE	60
<role definition>	CREATE ROLE	61
<rollback statement>	ROLLBACK WORK	62
<savepoint statement>	SAVEPOINT	63
<schema definition>	CREATE SCHEMA	64
<select statement: single row>	SELECT	65
<set connection statement>	SET CONNECTION	67
<set constraints mode statement>	SET CONSTRAINT	68
<set local time zone statement>	SET TIME ZONE	71
<set role statement>	SET ROLE	73
<set transaction statement>	SET TRANSACTION	75
<set session authorization identifier statement>	SET SESSION AUTHORIZATION	76
<SQL-server module definition>	DEFINE MODULE	51
<table definition>	CREATE TABLE	77
<test completion statement>	TEST COMPLETION	78
<translation definition>	CREATE TRANSLATION	79
<trigger definition>	CREATE TRIGGER	80
<update statement: positioned>	UPDATE CURSOR	81
<update statement: searched>	UPDATE WHERE	82
<view definition>	CREATE VIEW	84

Note: Value 85 is reserved for Part 3 of this Standard. Several other values are reserved for Part 5 of this Standard; please see the corresponding table in Part 5 for more information.

e) The value of ROW_COUNT is the number of rows affected as the result of executing a <delete statement: searched>, <insert statement>, or <update statement: searched> as a direct result of executing the previous SQL-statement. Let *S* be the <delete statement: searched>, <insert statement>, or <update statement: searched>. Let *T* be the table identified by the <table name> directly contained in *S*.

Case:

i) If *S* is an <insert statement>, then the value of ROW_COUNT is the number of rows inserted into *T*.

ii) If *S* is not an <insert statement> and does not contain a <search condition>, then the value of ROW_COUNT is the cardinality of *T* before the execution of *S*.

iii) Otherwise, let *SC* be the <search condition> directly contained in *S*. The value of ROW_COUNT is effectively derived by executing the statement:

SELECT COUNT(*) FROM *T* WHERE *SC*

before the execution of *S*.

The value of ROW_COUNT following the execution of an SQL-statement that does not directly result in the execution of a <delete statement: searched>, an <insert statement>, or an <update statement: searched> is implementation-dependent.

f) The value of TRANSACTIONS_COMMITTED is the number of SQL-transactions that have been committed since the most recent time at which the Diagnostics Area was emptied.

Note: See the General Rules of Subclause 12.4, "Rules for externally-invoked <routine>s".

g) The value of TRANSACTIONS_ROLLED_BACK is the number of SQL-transactions that have been rolled back since the most recent time at which the Diagnostics Area was emptied.

Note: See the General Rules of Subclause 12.4, "Rules for externally-invoked <routine>s".

h) The value of TRANSACTION_ACTIVE is 1 if an SQL-transaction is currently active, and 0 if one is not currently active.

2) If <condition information> was specified, then let *N* be the value of <condition number>. If *N* is less than 1 or greater than the number of conditions stored in the diagnostics area, then an exception condition is raised: *invalid condition number*. If <condition number> has the value 1, then the diagnostics information retrieved corresponds to the condition indicated by the SQLSTATE or SQLCODE value actually returned by execution of the previous SQL-statement other than a <get diagnostics statement>. Otherwise, the association between <condition number> values and specific conditions raised during evaluation of the General Rules for that SQL-statement is implementation-dependent.

3) Specification of <condition information item> retrieves information about the *N*-th condition in the diagnostics area into the <simple target specification>.

a) The value of CONDITION_NUMBER is the value of <condition number>.

b) The value of CLASS_ORIGIN is the identification of the naming authority that defined the class value of RETURNED_SQLSTATE. That value shall be 'ISO 9075' for any RETURNED_SQLSTATE whose class value is fully defined in Subclause 19.1, "SQLSTATE", and shall be an implementation-defined character string other than 'ISO 9075' for any RETURNED_SQLSTATE whose class value is an implementation-defined class value.

c) The value of SUBCLASS_ORIGIN is the identification of the naming authority that defined the subclass value of RETURNED_SQLSTATE. That value shall be 'ISO 9075' for any RETURNED_SQLSTATE whose subclass value is fully defined in Subclause 19.1, "SQLSTATE", and shall be an implementation-defined character string other than 'ISO 9075' for any RETURNED_SQLSTATE whose subclass value is an implementation-defined subclass value.

d) The value of RETURNED_SQLSTATE is the SQLSTATE parameter that would have been returned if this were the only completion or exception condition possible.

e) If the value of RETURNED_SQLSTATE corresponds to *warning* with a subclass of *cursor operation conflict*, then the value of CURSOR_NAME is the name of the cursor that caused the completion condition to be raised.

f) If the value of RETURNED_SQLSTATE corresponds to *integrity constraint violation*, *transaction rollback—integrity constraint violation*, or a *triggered data change violation* that was caused by a violation of a referential constraint, then:

i) The values of CONSTRAINT_CATALOG and CONSTRAINT_SCHEMA are the <catalog name> and the <unqualified schema name> of the <schema name> of the schema containing the constraint or assertion. The value of CONSTRAINT_NAME is the <qualified identifier> of the constraint or assertion.

ii) Case:

1) If the violated integrity constraint is a table constraint, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name>, and the <qualified identifier> or <local table name>, respectively, of the table in which the table constraint is contained.

2) If the violated integrity constraint is an assertion and if only one table referenced by the assertion has been modified as a result of executing the SQL-statement, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name>, and the <qualified identifier> or <local table name>, respectively, of the modified table.

3) Otherwise, the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are a zero-length string.

If TABLE_NAME identifies a declared local temporary table, then CATALOG_NAME is a zero-length string and SCHEMA_NAME is "MODULE".

g) If the value of RETURNED_SQLSTATE corresponds to *triggered action exception*, *transaction rollback—triggered action exception*, or a *triggered data change violation* that was caused by a trigger, then:

i) The values of TRIGGER_CATALOG and TRIGGER_SCHEMA are the <catalog name> and the <unqualified schema name> of the <schema name> of the schema containing the trigger. The value of TRIGGER_NAME is the <qualified identifier> of the <trigger name> of the trigger.

ii) The values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name>, and the <qualified identifier> of the <table name>, respectively, of the table on which the trigger is defined.

h) If the value of RETURNED_SQLSTATE corresponds to *syntax error or access rule violation*, then:

ANSI Only—caused by ISO changes not yet considered by ANSI

i) If the syntax error or access rule violation was caused by reference to a specific table, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name> of the schema that contains the table that caused the syntax error or access rule violation, and the <qualified identifier> or <local table name>, respectively. If TABLE_NAME refers to a declared local temporary table, then CATALOG_NAME is a zero-length string

and SCHEMA_NAME contains "MODULE". Otherwise, CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME contain a zero-length string.

ISO Only–SQL3

ii) Case:

1) If the syntax error or access rule violation was caused by reference to a specific table, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are:

A) If the specific table referenced was not a declared local temporary table, then the <catalog name>, the <unqualified schema name> of the <schema name> of the schema that contains the table that caused the syntax error or access rule violation, and the <qualified identifier>, respectively.

B) Otherwise, the a zero-length string, "MODULE", and the <local table name>, respectively.

2) Otherwise, CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME contain a zero-length string.

iii) If the syntax error or access rule violation was for an inaccessible column, then the value of COLUMN_NAME is the <column name> of that column. Otherwise, the value of COLUMN_NAME is a zero-length string.

i) If the value of RETURNED_SQLSTATE corresponds to *invalid cursor state*, then the value of CURSOR_NAME is the name of the cursor that is in the invalid state.

j) If the value of RETURNED_SQLSTATE corresponds to *with check option violation*, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are the <catalog name>, the <unqualified schema name> of the <schema name> of the schema that contains the view that caused the violation of the WITH CHECK OPTION, and the <qualified identifier> of that view, respectively.

k) If the value of RETURNED_SQLSTATE does not correspond to *syntax error or access rule violation*, then:

i) If the values of CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, and COLUMN_NAME identify a column for which no privileges are granted to the current <authorization identifier> or to the enabled applicable roles of that <authorization identifier>, then the value of COLUMN_NAME is replaced by a zero-length string.

ii) If the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME identify a table for which no privileges are granted to the current <authorization identifier> or to the enabled applicable roles of that <authorization identifier>, then the values of CATALOG_NAME, SCHEMA_NAME, and TABLE_NAME are replaced by a zero-length string.

iii) If the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME identify a <table constraint> for some table *T* and if no privileges for *T* are granted to the current <authorization identifier> or to the enabled applicable roles of that <authorization identifier>, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are replaced by a zero-length string.

iv) If the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME identify an assertion contained in some schema *S* and if the owner of *S* is not the <authorization identifier> of the current SQL-session, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are replaced by a zero-length string.

l) If the value of RETURNED_SQLSTATE corresponds to *external function call exception* or *external function exception*, then;

i) The values of ROUTINE_CATALOG and ROUTINE_SCHEMA are the <catalog name> and the <unqualified schema name>, respectively, of the <schema name> of the schema containing the routine.

ii) The values of ROUTINE_NAME and SPECIFIC_NAME are the <identifier> of the <routine name> and the <identifier> of the <specific name> of the routine, respectively.

m) If the value of RETURNED_SQLSTATE corresponds to *external function call exception*, *external function exception*, or *warning*, then the value of MESSAGE_TEXT is the message text item of the routine that raised the exception. Otherwise the value of MESSAGE_TEXT is an implementation-defined character string.

Note: An implementation may set this to <space>s, to a zero-length string, or to a character string describing the condition indicated by RETURNED_SQLSTATE.

n) The value of MESSAGE_LENGTH is the length in characters of the character string value in MESSAGE_TEXT.

o) The value of MESSAGE_OCTET_LENGTH is the length in octets of the character string value in MESSAGE_TEXT.

p) The values of CONNECTION_NAME and SERVER_NAME are respectively

Case:

i) If COMMAND_FUNCTION or DYNAMIC_FUNCTION identifies an <SQL connection statement>, then the <connection name> and the <SQL-server name> specified by or implied by the <SQL connection statement>.

ii) Otherwise, the <connection name> and <SQL-server name> of the SQL-session in which the condition was raised.

q) The values of character string items where not otherwise specified by the preceding rules are set to a zero-length string.

r) If the value of RETURNED_SQLSTATE corresponds to *data exception—numeric value out of range*, *data exception—invalid character value for case*, *data exception—string data, right trunctions*, *data exception—interval field overflow*, *data exception—invalid enumeration*

name, *integrity constraint violation*, *warning—string data, right truncation*, or *warning—implicit zero-bit padding*, then:

i) If the error occurred as a result of a cast to an input parameter or a cast from an output parameter during a routine invocation, and if a <parameter name> was specified for the parameter when the routine was created, then the value of PARAMETER_NAME is the <parameter name> of that parameter.

ii) Otherwise, the value of PARAMETER_NAME is a zero-length string.

Note: There are no numeric items that are not set by these rules.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not specify a <statement information item name> that is COMMAND_FUNCTION_CODE or DYNAMIC_FUNCTION_CODE.

   b) Conforming Full SQL language shall not specify a <statement information item name> that is TRIGGER_CATALOG, TRIGGER_SCHEMA, or TRIGGER_NAME.

   c) Conforming Full SQL language shall not specify a <statement information item name> that is ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME, or SPECIFIC_NAME.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any <get diagnostics statement>.

18 Information Schema and Definition Schema

18.1 Introduction

****Editor's Note****

Paper YOK-046R1/X3H2-93-185 noted that the Definition Schema and Information Schema do not have representations for the privileges "ALL SCHEMA PRIVILEGES", "UNDER", or "EXECUTE", which are defined in Subclause 10.4, "<privileges>". See Possible Problem 282 in the Editor's Notes.

The views of the Information Schema are viewed tables defined in terms of the base tables of the Definition Schema. The only purpose of the Definition Schema is to provide a data model to support the Information Schema and to assist understanding. An implementation need do no more than simulate the existence of the Definition Schema, as viewed through the Information Schema views.

The Information Schema views are defined as being in a schema named INFORMATION_SCHEMA, enabling these views to be accessed in the same way as any other tables in any other schema. SELECT on all of these views is granted to PUBLIC WITH GRANT OPTION, so that they can be queried by any user and so that SELECT privilege can be further granted on views that reference these Information Schema views. No other privilege is granted on them, so they cannot be updated.

The Information Schema also contains a small number of domains on which the columns of the Definition Schema are based. USAGE on all these domains is granted to PUBLIC WITH GRANT OPTION, so that they can be used by any user.

An implementation may define objects that are associated with INFORMATION_SCHEMA that are not defined in this Clause. An implementation may also add columns to tables that are defined in this Clause.

The Definition Schema base tables are defined as being in a schema named DEFINITION_SCHEMA. Because <unqualified schema name>s are prohibited from specifying DEFINITION_SCHEMA, the Definition Schema cannot be accessed in an SQL-statement.

Note: The Information Schema tables may be supposed to be represented in the Definition Schema in the same way as any other tables, and are hence self-describing.

Note: The Information Schema is a definition of the SQL data model, specified as an SQL-schema, in terms of <SQL schema statement>s as defined in this
ANSI American
ISO International
standard. Constraints defined in this Clause are not actual SQL constraints.

<identifier>s are represented in the tables of the Information Schema by the value of the <character string literal> corresponding to their <identifier body>s (in the case of <regular identifier>s) or their <delimited identifier body>s (in the case of <delimited identifier>s). The corresponding <character string literal> and the comparison of <identifier>s is defined in Subclause 5.2, "<token> and <separator>". Where an <identifier> has many equivalent forms, the one encountered at definition time is stored (of course, any lower case letters appearing in a <regular identifier> will have been converted to the corresponding upper case letter and any <doublequote symbol> appearing in a <delimited identifier> will have been replaced with a <duoble quote> before the <identifier> is stored in any table of the Information Schema).

Note: The *value* of a <character string literal> does not the <quote>s that surround a <character string literal> or the "<introducer><character set specification>" that can be specified with a <character string literal>.

****Editor's Note****

The redundancy in the Description of some of the Information Schema tables has been identified as a Possible Problem. See Possible Problem **091** in the Editor's Notes.

18.2 Information Schema

18.2.1 INFORMATION_SCHEMA Schema

Function

Identify the schema that is to contain the Information Schema tables.

Definition

```
CREATE SCHEMA INFORMATION_SCHEMA
       AUTHORIZATION INFORMATION_SCHEMA
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.2 INFORMATION_SCHEMA_CATALOG_NAME base table

Function

Identify the catalog that contains the Information Schema.

Definition

```
CREATE TABLE INFORMATION_SCHEMA_CATALOG_NAME
       ( CATALOG_NAME        SQL_IDENTIFIER,
         CONSTRAINT INFORMATION_SCHEMA_CATALOG_NAME_PRIMARY_KEY
           PRIMARY KEY ( CATALOG_NAME ),
         CONSTRAINT INFORMATION_SCHEMA_CATALOG_NAME_CHECK
           CHECK ( ( SELECT COUNT(*) FROM INFORMATION_SCHEMA_CATALOG_NAME ) = 1 )
        )
```

Description

1) The value of CATALOG_NAME is the name of the catalog in which this Information Schema resides.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.3 INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY assertion

Function

Ensure that there is exactly one row in the INFORMATION_SCHEMA_CATALOG_NAME table.

Definition

```
CREATE ASSERTION INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY
       CHECK ( 1 = ( SELECT COUNT(*)
                       FROM INFORMATION_SCHEMA_CATALOG_NAME ) )
```

Description

1) The INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY assertion ensures that there is exactly one row in the INFORMATION_SCHEMA_CATALOG_NAME table.

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.4 SCHEMATA view

Function

Identify the schemata that are owned by a given user.

Definition

```
CREATE VIEW SCHEMATA
       AS SELECT
           CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER,
           DEFAULT_CHARACTER_SET_CATALOG, DEFAULT_CHARACTER_SET_SCHEMA,
           DEFAULT_CHARACTER_SET_NAME
         FROM DEFINITION_SCHEMA.SCHEMATA
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND CATALOG_NAME
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.5 DOMAINS view

Function

Identify the domains defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW DOMAINS
       AS SELECT DISTINCT
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
         DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
         COLLATION_CATALOG, COLLATION_SCHEMA,COLLATION_NAME,
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE,
         DATETIME_PRECISION, INTERVAL_CODE, INTERVAL_PRECISION, DOMAIN_DEFAULT
         ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
         ABSTRACT_DATA_TYPE_NAME,
         NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME
       FROM DEFINITION_SCHEMA.DOMAINS
            JOIN
              DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D
              LEFT JOIN
              DEFINITION_SCHEMA.COLLATIONS AS S
              USING ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
            ON
              ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, '' )
              = ( TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                    TABLE_OR_DOMAIN_NAME, COLUMN_NAME ) )
         WHERE
                ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, 'DOMAIN' )
                  IN
                  ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                    FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                      WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
                OR
                ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IN
                  ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
                      FROM COLUMNS ) )
           AND DOMAIN_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.6 DOMAIN_CONSTRAINTS view

Function

Identify the domain constraints of domains in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW DOMAIN_CONSTRAINTS
       AS SELECT DISTINCT
         CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
         IS_DEFERRABLE, INITIALLY_DEFERRED
       FROM DEFINITION_SCHEMA.DOMAIN_CONSTRAINTS
            JOIN
              DEFINITION_SCHEMA.SCHEMATA AS S
                 ON
                 ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
                 = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE
           SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.7 TABLES view

Function

Identify the tables defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW TABLES
       AS SELECT
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, TABLE_TYPE
       FROM DEFINITION_SCHEMA.TABLES
           WHERE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
             IN (
                 SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                  FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
                    WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                 UNION
                 SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                  FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
                    WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.8 VIEWS view

Function

Identify the viewed tables defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW VIEWS
       AS SELECT
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
         CASE WHEN ( TABLE_CATALOG, TABLE_SCHEMA, CURRENT_USER )
                     IN ( SELECT CATALOG_NAME, SCHEMA_NAME, SCHEMA_OWNER
                                 FROM DEFINITION_SCHEMA.SCHEMATA )
              THEN VIEW_DEFINITION
              ELSE NULL
         END AS VIEW_DEFINITION,
           CHECK_OPTION, IS_UPDATABLE
       FROM DEFINITION_SCHEMA.VIEWS
           WHERE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
             IN ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                  FROM TABLES )
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.9 COLUMNS view

Function

Identify the columns of tables defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW COLUMNS
       AS SELECT DISTINCT
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
         C.COLUMN_NAME, ORDINAL_POSITION,
         CASEWHEN EXISTS ( SELECT *
                FROM DEFINITION_SCHEMA.SCHEMATA AS S
                WHERE ( TABLE_CATALOG, TABLE_SCHEMA )
                    = (S.CATALOG_NAME, S.SCHEMA_NAME )
                  AND SCHEMA_OWNER = USER )
               THEN COLUMN_DEFAULT
              ELSE NULL
         END AS COLUMN_DEFAULT,
         IS_NULLABLE,
         COALESCE (D1.DATA_TYPE, D2.DATA_TYPE) AS DATA_TYPE,
         COALESCE (D1.CHARACTER_MAXIMUM_LENGTH, D2.CHARACTER_MAXIMUM_LENGTH)
           AS CHARACTER_MAXIMUM_LENGTH,
         COALESCE (D1.CHARACTER_OCTET_LENGTH, D2.CHARACTER_OCTET_LENGTH)
                     AS CHARACTER_OCTET_LENGTH,
         COALESCE (D1.NUMERIC_PRECISION, D2.NUMERIC_PRECISION)
                     AS NUMERIC_PRECISION,
         COALESCE (D1.NUMERIC_PRECISION_RADIX, D2.NUMERIC_PRECISION_RADIX)
                     AS NUMERIC_PRECISION_RADIX,
         COALESCE (D1.NUMERIC_SCALE, D2.NUMERIC_SCALE) AS NUMERIC_SCALE,
         COALESCE (D1.DATETIME_PRECISION, D2.DATETIME_PRECISION) AS DATETIME_PRECISION,
         COALESCE (D1.INTERVAL_CODE, D3.INTERVAL_CODE) AS INTERVAL_CODE,
         COALESCE (D1.INTERVAL_PRECISION, D3.INTERVAL_PRECISION) AS INTERVAL_PRECISION,
         COALESCE (C1.CHARACTER_SET_CATALOG, C2.CHARACTER_SET_CATALOG)
                     AS CHARACTER_SET_CATALOG,
         COALESCE (C1.CHARACTER_SET_SCHEMA, C2.CHARACTER_SET_SCHEMA)
                     AS CHARACTER_SET_SCHEMA,
         COALESCE (C1.CHARACTER_SET_NAME, C2.CHARACTER_SET_NAME) AS CHARACTER_SET_NAME,
         COALESCE (D1.COLLATION_CATALOG, D2.COLLATION_CATALOG) AS COLLATION_CATALOG,
         COALESCE (D1.COLLATION_SCHEMA, D2.COLLATION_SCHEMA) AS COLLATION_SCHEMA,
         COALESCE (D1.COLLATION_NAME, D2.COLLATION_NAME) AS COLLATION_NAME,
         DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
         COALESCE (D1.ABSTRACT_DATA_TYPE_CATALOG, D2.ABSTRACT_DATA_TYPE_CATALOG)
                  AS ABSTRACT_DATA_TYPE_CATALOG,
         COALESCE (D1.ABSTRACT_DATA_TYPE_SCHEMA, D2.ABSTRACT_DATA_TYPE_SCHEMA)
                  AS ABSTRACT_DATA_TYPE_SCHEMA,
         COALESCE (D1.ABSTRACT_DATA_TYPE_NAME, D2.ABSTRACT_DATA_TYPE_NAME)
                  AS ABSTRACT_DATA_TYPE_NAME,
         COALESCE (D1.NULL_CLASS_CATALOG, D2.NULL_CLASS_CATALOG) AS NULL_CLASS_CATALOG,
         COALESCE (D1.NULL_CLASS_SCHEMA, D2.NULL_CLASS_SCHEMA) AS NULL_CLASS_SCHEMA,
         COALESCE (D1.NULL_CLASS_NAME, D2.NULL_CLASS_NAME) AS NULL_CLASS_NAME
       FROM DEFINITION_SCHEMA.COLUMNS AS C
           LEFT JOIN
             DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
               LEFT JOIN
                 DEFINITION_SCHEMA.COLLATIONS AS C1
                 ON
                 ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA, C1.COLLATION_NAME )
                 = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA, D1.COLLATION_NAME ) )
           ON
             ( ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME, C.COLUMN_NAME )
```

```
    = ( D1.TABLE_OR_DOMAIN_CATALOG, D1.TABLE_OR_DOMAIN_SCHEMA,
          D1.TABLE_OR_DOMAIN_NAME, D1.COLUMN_NAME ) )
LEFT JOIN
  DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D2
    LEFT JOIN
      DEFINITION_SCHEMA.COLLATIONS AS C2
      ON
      ( ( C2.COLLATION_CATALOG, C2.COLLATION_SCHEMA, C2.COLLATION_NAME )
      = ( D2.COLLATION_CATALOG, D2.COLLATION_SCHEMA, D2.COLLATION_NAME ) )
ON
  ( ( C.DOMAIN_CATALOG, C.DOMAIN_SCHEMA, C.DOMAIN_NAME )
  = ( D2.TABLE_OR_DOMAIN_CATALOG, D2.TABLE_OR_DOMAIN_SCHEMA,
          D2.TABLE_OR_DOMAIN_NAME ) )
WHERE ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME, C.COLUMN_NAME )
  IN
    ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
        FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
          WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
AND C.TABLE_CATALOG
  = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.10 ABSTRACT_DATA_TYPES view

Function

Identify the abstract data types defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW ABSTRACT_DATA_TYPES AS
       SELECT
           ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
           ABSTRACT_DATA_TYPE_NAME, ABSTRACT_DATA_TYPE_EQUALS,
           ABSTRACT_DATA_TYPE_LESS_THAN
         FROM DEFINITION_SCHEMA.ABSTRACT_DATA_TYPES
         WHERE ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                 ABSTRACT_DATA_TYPE_NAME ) IN
               ( SELECT ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                        ABSTRACT_DATA_TYPE_NAME
                   FROM DEFINITION_SCHEMA.ABSTRACT_DATA_TYPE_PRIVILEGES
                   WHERE GRANTEE IN ( 'PUBLIC', USER ) )
           AND
               ABSTRACT_DATA_TYPE_CATALOG
             = ( SELECT CATALOG_NAME
                   FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

****Editor's Note****

This view refers to the ABSTRACT_DATA_TYPE_OPERATIONS base table (see the FROM clause), but that base table doesn't seem to exist!

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.11 NULL_CLASSES view

Function

Identify the null classes that are accessible to a given user.

Definition

```
CREATE VIEW NULL_CLASSES AS
       SELECT DISTINCT
           NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME
         FROM DEFINITION_SCHEMA.NULL_CLASSES
         WHERE NULL_CLASS_SCHEMA = USER
            OR
               ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA,
                 NULL_CLASS_NAME, 'NULL_CLASS' ) IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE IN ('PUBLIC', USER ) )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.12 NULL_STATES view

Function

Identify the null states that are accessible to a given user.

Definition

```
CREATE VIEW NULL_STATES
       AS SELECT DISTINCT
         NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME
         NULL_STATE_NAME, ORDINAL_POSITION
       FROM DEFINITION_SCHEMA.NULL_STATES
         WHERE NULL_CLASS_SCHEMA = USER
               OR
               ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA,
                 NULL_CLASS_NAME, 'NULL_CLASS' ) IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE IN ('PUBLIC', USER ) )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.13 TABLE_PRIVILEGES view

Function

Identify the privileges on tables defined in this catalog that are available to or granted by a given user.

Definition

```
CREATE VIEW TABLE_PRIVILEGES AS
       SELECT
           GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
           PRIVILEGE_TYPE, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
         WHERE
           ( GRANTEE IN ( 'PUBLIC', CURRENT_USER )
             OR GRANTOR = CURRENT_USER )
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.14 COLUMN_PRIVILEGES view

Function

Identify the privileges on columns of tables defined in this catalog that are available to or granted by a given user.

Definition

```
CREATE VIEW COLUMN_PRIVILEGES AS
       SELECT
           GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
           PRIVILEGE_TYPE, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
         WHERE
           ( GRANTEE IN ( 'PUBLIC', CURRENT_USER )
           OR GRANTOR = CURRENT_USER )
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.15 USAGE_PRIVILEGES view

Function

Identify the USAGE privileges on objects defined in this catalog that are available to or granted by a given user.

Definition

```
CREATE VIEW USAGE_PRIVILEGES AS
       SELECT
           GRANTOR, GRANTEE, OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
           OBJECT_TYPE, 'USAGE' AS PRIVILEGE_TYPE, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
         WHERE
           GRANTEE IN ( 'PUBLIC', CURRENT_USER )
           OR GRANTOR = CURRENT_USER )
           AND OBJECT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.16 ABSTRACT_DATA_TYPE_PRIVILEGES view

Function

Identify the privileges on abstract data types defined in this catalog that are accessible to or granted by a given user.

Definition

```
CREATE VIEW ABSTRACT_DATA_TYPE_PRIVILEGES AS
       SELECT
           GRANTOR, GRANTEE, ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
           ABSTRACT_DATA_TYPE_NAME, PRIVILEGE_TYPE, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.ABSTRACT_DATA_TYPE_PRIVILEGES
         WHERE ( GRANTEE IN ( 'PUBLIC', USER )
            OR GRANTOR = USER )
           AND TABLE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.17 COLUMN_ABSTRACT_DATA_TYPE_USAGE view

Function

Identify the columns defined that are dependent on an abstract data type defined in this catalog and owned by a given user.

Definition

```
CREATE VIEW ABSTRACT_DATA_TYPE_IMPACT_COLUMN AS
       SELECT
           ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
           ABSTRACT_DATA_TYPE_NAME,
           TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
         FROM
           DEFINITION_SCHEMA.COLUMNS C
           JOIN
           DEFINITION_SCHEMA.SCHEMATA S
             ON ( C.ABSTRACT_DATA_TYPE_CATALOG, C.ABSTRACT_DATA_TYPE_SCHEMA )
              = ( S.CATALOG_NAME, S.SCHEMA_NAME )
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND C.ABSTRACT_DATA_TYPE_NAME IS NOT NULL
           AND C.ABSTRACT_DATA_TYPE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.18 DOMAIN_ABSTRACT_DATA_TYPE_USAGE view

Function

Identify the domains defined that are dependent on abstract data types defined in this catalog and owned by a given user.

Definition

```
CREATE VIEW ABSTRACT_DATA_TYPE_IMPACT_DOMAIN AS
       SELECT
           ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
           ABSTRACT_DATA_TYPE_NAME,
           DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
         FROM DEFINITION_SCHEMA.DOMAINS D
           JOIN
           DEFINITION_SCHEMA.SCHEMATA S
             ON ( D.ABSTRACT_DATA_TYPE_CATALOG, D.ABSTRACT_DATA_TYPE_SCHEMA )
              = ( S.CATALOG_NAME, S.SCHEMA_NAME )
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND C.ABSTRACT_DATA_TYPE_NAME IS NOT NULL
           AND C.ABSTRACT_DATA_TYPE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.19 TABLE_CONSTRAINTS view

Function

Identify the table constraints defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW TABLE_CONSTRAINTS
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
           TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
           CONSTRAINT_TYPE, IS_DEFERRABLE, INITIALLY_DEFERRED
         FROM DEFINITION_SCHEMA.TABLE_CONSTRAINTS
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.20 REFERENTIAL_CONSTRAINTS view

Function

Identify the referential constraints defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW REFERENTIAL_CONSTRAINTS
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
           UNIQUE_CONSTRAINT_CATALOG, UNIQUE_CONSTRAINT_SCHEMA, UNIQUE_CONSTRAINT_NAME,

           MATCH_OPTION, UPDATE_RULE, DELETE_RULE
         FROM DEFINITION_SCHEMA.REFERENTIAL_CONSTRAINTS
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.21 CHECK_CONSTRAINTS view

Function

Identify the check constraints defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW CHECK_CONSTRAINTS
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME, CHECK_CLAUSE
         FROM DEFINITION_SCHEMA.CHECK_CONSTRAINTS
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.22 KEY_COLUMN_USAGE view

Function

Identify the columns defined in this catalog that are constrained as keys by a given user.

Definition

```
CREATE VIEW KEY_COLUMN_USAGE
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
           TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, ORDINAL_POSITION
         FROM DEFINITION_SCHEMA.KEY_COLUMN_USAGE
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.23 ASSERTIONS view

Function

Identify the assertions defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW ASSERTIONS
       AS SELECT
           CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
           IS_DEFERRABLE, INITIALLY_DEFERRED
         FROM DEFINITION_SCHEMA.ASSERTIONS
              JOIN
              DEFINITION_SCHEMA.SCHEMATA S
              ON
               ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.24 CHARACTER_SETS view

Function

Identify the character sets defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW CHARACTER_SETS
       AS SELECT
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         FORM_OF_USE, NUMBER_OF_CHARACTERS,
         DEFAULT_COLLATE_CATALOG, DEFAULT_COLLATE_SCHEMA, DEFAULT_COLLATE_NAME
       FROM DEFINITION_SCHEMA.CHARACTER_SETS
       WHERE ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
               CHARACTER_SET_NAME, 'CHARACTER SET')
               IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
           AND CHARACTER_SET_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.25 COLLATIONS view

Function

Identify the character collations defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW COLLATIONS
       AS SELECT
         COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME,
         CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
         PAD_ATTRIBUTE,
           COLLATION_TYPE, COLLATION_DEFINITION, COLLATION_DICTIONARY
       FROM DEFINITION_SCHEMA.COLLATIONS
       WHERE ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, 'COLLATION')
               IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
           AND COLLATION_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not reference the COLLATIONS view.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.26 TRANSLATIONS view

Function

Identify the character translations defined in this catalog that are accessible to a given user.

Definition

```
CREATE VIEW TRANSLATIONS
       AS SELECT
         TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME,
         SOURCE_CHARACTER_SET_CATALOG, SOURCE_CHARACTER_SET_SCHEMA,
           SOURCE_CHARACTER_SET_NAME,
         TARGET_CHARACTER_SET_CATALOG, TARGET_CHARACTER_SET_SCHEMA,
           TARGET_CHARACTER_SET_NAME
       FROM DEFINITION_SCHEMA.TRANSLATIONS
       WHERE ( TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME, 'TRANSLATION')

               IN
               ( SELECT OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE
                   FROM DEFINITION_SCHEMA.USAGE_PRIVILEGES
                   WHERE GRANTEE IN ( 'PUBLIC', CURRENT_USER ) )
           AND TRANSLATION_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   a) Conforming Intermediate SQL language shall not reference the TRANSLATIONS view.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.27 VIEW_TABLE_USAGE view

Function

Identify the tables on which viewed tables defined in this catalog and owned by a given user are dependent.

Definition

```
CREATE VIEW VIEW_TABLE_USAGE
       AS SELECT
         VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME,
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
       FROM DEFINITION_SCHEMA.VIEW_TABLE_USAGE
            JOIN
            DEFINITION_SCHEMA.SCHEMATA S
           ON
             ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND VIEW_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.28 VIEW_COLUMN_USAGE view

Function

Identify the columns on which viewed tables defined in this catalog and owned by a given user are dependent.

Definition

```
CREATE VIEW VIEW_COLUMN_USAGE AS
       SELECT VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME,
              TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
         FROM DEFINITION_SCHEMA.VIEW_COLUMN_USAGE
           JOIN
           DEFINITION_SCHEMA.SCHEMATA S
             ON
             ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND VIEW_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.29 CONSTRAINT_TABLE_USAGE view

Function

Identify the tables that are used by referential constraints, unique constraints, check constraints, and assertions defined in this catalog and owned by a given user.

Definition

```
CREATE VIEW CONSTRAINT_TABLE_USAGE
       AS
         SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME FROM
       (
         ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                  CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
             FROM DEFINITION_SCHEMA.CHECK_COLUMN_USAGE )
       UNION
         ( SELECT PK.TABLE_CATALOG, PK.TABLE_SCHEMA, PK.TABLE_NAME,
                  FK.CONSTRAINT_CATALOG, FK.CONSTRAINT_SCHEMA, FK.CONSTRAINT_NAME
             FROM
             DEFINITION_SCHEMA.REFERENTIAL_CONSTRAINTS AS FK
             JOIN
             DEFINITION_SCHEMA.TABLE_CONSTRAINTS AS PK
               ON
                 ( FK.UNIQUE_CONSTRAINT_CATALOG, FK.UNIQUE_CONSTRAINT_SCHEMA,
                     FK.UNIQUE_CONSTRAINT_NAME )
               = ( PK.CONSTRAINT_CATALOG, PK.CONSTRAINT_SCHEMA, PK.CONSTRAINT_NAME )
         )
       )
       JOIN
         DEFINITION_SCHEMA.SCHEMATA S
            ON
             ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE S.SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.30 CONSTRAINT_COLUMN_USAGE view

Function

Identify the columns used by referential constraints, unique constraints, check constraints, and assertions defined in this catalog and owned by a given user.

Definition

```
CREATE VIEW CONSTRAINT_COLUMN_USAGE
       AS
         SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
                CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME FROM
           ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME,
                  CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
             FROM DEFINITION_SCHEMA.CHECK_COLUMN_USAGE )
           UNION
           ( SELECT K.TABLE_CATALOG, K.TABLE_SCHEMA, K.TABLE_NAME, K.COLUMN_NAME,
                    CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
               FROM DEFINITION_SCHEMA.TABLE_CONSTRAINTS
             JOIN DEFINITION_SCHEMA.KEY_COLUMN_USAGE AS K
               USING ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
         )
         JOIN
           DEFINITION_SCHEMA.SCHEMATA
            ON
             ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( CATALOG_NAME, SCHEMA_NAME ) )
           WHERE SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.31 COLUMN_DOMAIN_USAGE view

Function

Identify the columns defined that are dependent on a domain defined in this catalog and owned by a user.

Definition

```
CREATE VIEW COLUMN_DOMAIN_USAGE
       AS SELECT
         D.DOMAIN_CATALOG, D.DOMAIN_SCHEMA, D.DOMAIN_NAME,
         TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
       FROM DEFINITION_SCHEMA.COLUMNS C
            JOIN
                DEFINITION_SCHEMA.DOMAINS D
                JOIN
                DEFINITION_SCHEMA.SCHEMATA S
                  ON ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA )
                  = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
            ON ( ( D.DOMAIN_CATALOG, D.DOMAIN_SCHEMA, D.DOMAIN_NAME )
                  = ( C.DOMAIN_CATALOG, C.DOMAIN_SCHEMA, C.DOMAIN_NAME ) )
         WHERE SCHEMA_OWNER = CURRENT_USER
           AND C.DOMAIN_NAME IS NOT NULL
           AND D.DOMAIN_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.2.32 OPERATORS view

Function

Identify the operators that are accessible in the SQL-environment.

Definition

```
CREATE VIEW OPERATORS AS
       SELECT
           OPERATOR_CATALOG, OPERATOR_SCHEMA, OPERATOR_NAME, OPERATOR_TYPE,
           OPERATOR_LEVEL, OPERATOR_FORM
         FROM DEFINITION_SCHEMA.OPERATORS
         WHERE CATALOG_NAME =
               ( SELECT CATALOG_NAME
                   FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the OPERATORS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.33 TRIGGERS view

Function

Identify the triggers in this catalog that are owned by a given user.

Definition

```
CREATE VIEW TRIGGERS
       AS SELECT
         TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME, EVENT_MANIPULATION,
         EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE,
         EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE
         CONDITION_TIMING, CONDITION_REFERENCE_OLD_TABLE, CONDITION_REFERENCE_NEW_TABLE,

         ACTION_ORDER, ACTION_CONDITION, ACTION_STATEMENT_LIST, ACTION_ORIENTATION,
         COLUMN_LIST_IS_IMPLICIT
       FROM DEFINITION_SCHEMA.TRIGGERS
         JOIN DEFINITION_SCHEMA.SCHEMATA S
         ON
           ( ( TRIGGER_CATALOG, TRIGGER_SCHEMA )
           =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = USER
           AND
               ROUTINE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the TRIGGERS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.34 TRIGGERED_COLUMNS view

Function

Identify the columns that are referenced by a trigger defined in this catalog that are owned by a given user.

Definition

```
CREATE VIEW TRIGGERED_COLUMNS
       AS SELECT
         TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME,
         EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE,
         EVENT_OBJECT_COLUMN,
       FROM DEFINITION_SCHEMA.TRIGGERED_COLUMNS
         JOIN DEFINITION_SCHEMA.SCHEMATA S
         ON
           ( ( TRIGGER_CATALOG, TRIGGER_SCHEMA )
           =
               ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
         WHERE SCHEMA_OWNER = USER
           AND
               ROUTINE_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the TRIGGERED_COLUMNS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.35 ENABLED_APPLICABLE_ROLES view

Function

Identify the enabled applicable roles for the current SQL-session.

Definition

```
CREATE VIEW ENABLED_APPLICABLE_ROLES ( ROLE_NAME )
       AS
         ( SELECT ENABLED_ROLE FROM USER_SESSION_ENABLED_ROLE
           RECURSIVE UNION R
           SELECT ROLE_NAME FROM ROLE_AUTHORIZATION_DESCRIPTORS RAD
             WHERE RAD.GRANTEE = R.ROLE_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the ENABLED_APPLICABLE_ROLES view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.36 ROLE_TABLE_GRANTS view

Function

Identifies the privileges on tables defined in this catalog that are available to the currently enabled applicable roles.

Definition

```
CREATE VIEW ROLE_TABLE_GRANTS AS
       SELECT GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
           PRIVILEGE_TYPE
         FROM DEFINITION_SCHEMA.TABLE_PRIVILEGES
         WHERE GRANTEE IN
                 ( SELECT ENABLED_ROLE FROM USER_SESSION_ENABLED_ROLE
                   RECURSIVE UNION R
                   SELECT ROLE_NAME
                     FROM DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                     WHERE RAD.GRANTEE = R.ROLE_NAME ) )
               AND TABLE_CATALOG
                 = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the ROLE_TABLE_GRANTS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.37 ROLE_COLUMN_GRANTS view

Function

Identifies the privileges on columns defined in this catalog that are available to the currently enabled applicable roles.

Definition

```
CREATE  VIEW ROLE_COLUMN_GRANTS AS
       SELECT GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
           COLUMN_NAME, PRIVILEGE_TYPE
         FROM DEFINITION_SCHEMA.COLUMN_PRIVILEGES
         WHERE GRANTEE IN
                       ( SELECT ENABLED_ROLE FROM USER_SESSION_ENABLED_ROLE
                         RECURSIVE UNION R
                         SELECT ROLE_NAME FROM
                           DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                               WHERE RAD.GRANTEE = R.ROLE_NAME ) )
           AND
               TABLE_CATALOG =
               ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the ROLE_COLUMN_GRANTS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.38 APPLICABLE_ROLES view

Function

Identifies the applicable roles for the current user.

Definition

```
CREATE  VIEW APPLICABLE_ROLES AS
       SELECT DISTINCT GRANTEE, ROLE_NAME, IS_GRANTABLE
         FROM ( SELECT GRANTEE, ROLE_NAME, IS_GRANTABLE
                  FROM DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS
                  WHERE GRANTEE IN ( USER, 'PUBLIC' )
                RECURSIVE UNION R
                SELECT GRANTEE, ROLE_NAME
                  FROM DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS RAD
                  WHERE RAD.GRANTEE = R.ROLE_NAME ) )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the APPLICABLE_ROLES view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.2.39 ADMINISTRABLE_ROLE_AUTHORIZATIONS view

Function

Identify role authorizations for which the current user has WITH
ANSI GRANT
ISO ADMIN
OPTION.

Definition

```
CREATE VIEW ADMINISTRABLE_ROLE_AUTHORIZATIONS AS
       SELECT GRANTEE, ROLE_NAME, IS_GRANTABLE
         FROM DEFINITION_SCHEMA.ROLE_AUTHORIZATION_DESCRIPTORS
         WHERE ROLE_NAME IN
                       ( SELECT ROLE_NAME
                           FROM APPLICABLE_ROLES
                           WHERE IS_GRANTABLE = 'YES' )
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the ADMINISTRABLE_ROLE_AUTHORIZATIONS view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.3 USER_SESSION_ENABLED_ROLE view

Function

Identify the current authorization identifier and the role set by a SET ROLE statement for the current SQL-session.

Definition

```
CREATE VIEW USER_SESSION_ENABLED_ROLE ( USER_NAME, ENABLED_ROLE )
       AS SELECT USER_NAME, ENABLED_ROLE
       FROM DEFINITION_SCHEMA.USER_SESSION_ENABLED_ROLE
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall not reference the USER_SESSION_ENABLED_ROLE view.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18.3.1 SQL_LANGUAGES view

Function

Identify the conformance levels, options, and dialects supported by the SQL-implementation processing data defined in this catalog.

Definition

```
CREATE VIEW SQL_LANGUAGES
       AS SELECT
         SQL_LANGUAGE_SOURCE, SQL_LANGUAGE_YEAR, SQL_LANGUAGE_CONFORMANCE,
           SQL_LANGUAGE_INTEGRITY, SQL_LANGUAGE_IMPLEMENTATION,
           SQL_LANGUAGE_BINDING_STYLE, SQL_LANGUAGE_PROGRAMMING_LANGUAGE
       FROM DEFINITION_SCHEMA.SQL_LANGUAGES
```

Leveling Rules

1) The following restrictions apply for Full SQL:

   None.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not reference the Information Schema.

18.3.2 SQL_IDENTIFIER domain

Function

Define a domain that contains all valid <identifier>s.

Definition

```
CREATE DOMAIN SQL_IDENTIFIER AS CHARACTER VARYING (L)
       CHARACTER SET SQL_TEXT
```

Description

1) This domain specifies any variable-length character value that conforms to the rules for an SQL <identifier>.

   Note: There is no way in SQL to specify a <domain constraint> that would be true for any valid SQL <identifier> and false for all other values.

2) *L* is the implementation-defined maximum length of <identifier>.

18.3.3 CHARACTER_DATA domain

Function

Define a domain that contains any character data.

Definition

```
CREATE DOMAIN CHARACTER_DATA AS CHARACTER VARYING (ML)
       CHARACTER SET SQL_TEXT
```

Description

1) This domain specifies any character data.

2) *ML* is the implementation-defined maximum length of a variable-length character string.

18.3.4 CARDINAL_NUMBER domain

Function

Define a domain that contains a non-negative number.

Definition

```
CREATE DOMAIN CARDINAL_NUMBER AS INTEGER
       CONSTRAINT CARDINAL_NUMBER_DOMAIN_CHECK CHECK ( VALUE >= 0 )
```

Description

1) The domain CARDINAL_NUMBER contains any non-negative number that is less than the implementation-defined maximum for INTEGER (i.e., the implementation-defined value of NUMERIC_PRECISION_RADIX raised to the power of implementation-defined NUMERIC_PRECISION).

18.4 Definition Schema

18.4.1 Introduction

The base tables of the Definition Schema are all defined in a <schema definition> for the schema named named DEFINITION_SCHEMA. The table definitions are as as complete as the definitional power of SQL allows. The table definitions are supplemented with assertions where appropriate; see Subclause 18.4.39, "Assertions on the base tables". Each description comprises three parts:

1) The function of the definition is stated.

2) The SQL definition of the object is presented as a <table definition>.

3) An explanation of the object.

The specification provides only a model of the base tables that are required, and does not imply that an implementation shall provide the functionality in the manner described in this clause.

An instance of a definition schema describes an instance of a cluster of catalogs (see Subclause 4.26, "Clusters of catalogs").

18.4.2 DEFINITION_SCHEMA Schema

Function

Create the schema that is to contain the base tables that underlie the Information Schema

Definition

```
CREATE SCHEMA DEFINITION_SCHEMA
      AUTHORIZATION DEFINITION_SCHEMA
```

Description

None.

18.4.3 USERS base table

Function

The USERS table has one row for each <authorization identifier> referenced in the Information Schema. These are all those <authorization identifier>s that may grant or receive privileges as well as those that may create a schema, or currently own a schema created through a <schema definition>.

Definition

```
CREATE TABLE USERS
       (
       USER_NAME     INFORMATION_SCHEMA.SQL_IDENTIFIER
           CONSTRAINT USERS_PRIMARY_KEY PRIMARY KEY
         CONSTRAINT USERS_CHECK CHECK ( USER_NAME NOT IN
                     ( SELECT ROLE_NAME FROM ROLES ) )
       )
```

Description

1) The values of USER_NAME are <authorization identifier>s that are known.

18.4.4 ROLES base table

Function

The ROLES table has one row for each <role name> for each role known to the database management system.

Definition

```
CREATE TABLE ROLES
       (
       ROLE_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER       PRIMARY KEY,
         CONSTRAINT ROLES_CHECK CHECK ( ROLE_NAME NOT IN
                     ( SELECT USER_NAME FROM USERS ) )
       )
```

Description

1) A row is inserted into this table each time a <role definition> is executed. A row is deleted from this table each time the <drop role statement> is executed.

2) The value of ROLE_NAME is the <role name> defined by <role definition>.

18.4.5 SCHEMATA base table

Function

The SCHEMATA table has one row for each schema.

Definition

```
CREATE TABLE SCHEMATA
       (
       CATALOG_NAME                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SCHEMA_NAME                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SCHEMA_OWNER                   INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT SCHEMA_OWNER_NOT_NULL NOT NULL,
       DEFAULT_CHARACTER_SET_CATALOG  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DEFAULT_CHARACTER_SET_SCHEMA   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DEFAULT_CHARACTER_SET_NAME     INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT SCHEMATA_PRIMARY_KEY PRIMARY KEY ( CATALOG_NAME, SCHEMA_NAME ),

       CONSTRAINT SCHEMATA_FOREIGN_KEY FOREIGN KEY ( SCHEMA_OWNER )
         REFERENCES USERS
       )
```

Description

1) All the values of CATALOG_NAME are the name of the catalog in which the schemata are included.

2) The values of SCHEMA_NAME are the unqualified schema names of the schemata in the catalog.

3) The values of SCHEMA_OWNER are the authorization identifiers that own the schemata.

4) The values of DEFAULT_CHARACTER_SET_CATALOG, DEFAULT_CHARACTER_SET_SCHEMA, and DEFAULT_CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the default character set for columns and domains in the schemata.

18.4.6 ABSTRACT_DATA_TYPES base table

Function

The ABSTRACT_DATA_TYPES table has one row for each abstract data type.

Definition

```
CREATE TABLE ABSTRACT_DATA_TYPES
       (
         ABSTRACT_DATA_TYPE_CATALOG              INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_SCHEMA               INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_NAME                 INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_EQUALS               INFORMATION_SCHEMA.CHARACTER_DATA,
         ABSTRACT_DATA_TYPE_LESS_THAN            INFORMATION_SCHEMA.CHARACTER_DATA,

  CONSTRAINT ABSTRACT_DATA_TYPES_PRIMARY_KEY
           PRIMARY KEY (ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                        ABSTRACT_DATA_TYPE_NAME),

  CONSTRAINT ABSTRACT_DATA_TYPES_FOREIGN_KEY
           FOREIGN KEY (ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA)
             REFERENCES SCHEMATA
       )
```

Description

1) A row is inserted in this table whenever an <abstract data type definition> is executed and a row is deleted whenever a <drop data type statement> is executed.

2) The values of ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA and ABSTRACT_DATA_TYPE_NAME are the qualified name of the abstract data type that is defined.

- 1 Description deleted.

3) The values of ABSTRACT_DATA_TYPE_EQUALS and ABSTRACT_DATA_TYPE_LESS_THAN are respectively the <external function name>s specified in the <comparisons clause> of the associated <abstract data type definition>.

18.4.7 DATA_TYPE_DESCRIPTOR base table

Function

The DATA_TYPE_DESCRIPTOR table has one row for each domain and one row for each column (in each table) that is defined as having a data type rather than a domain. It effectively contains a representation of the data type descriptors.

Definition

```
CREATE TABLE DATA_TYPE_DESCRIPTOR
       (
       TABLE_OR_DOMAIN_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_OR_DOMAIN_SCHEMA             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_OR_DOMAIN_NAME               INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME                        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DATA_TYPE                          INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_OR_DOMAIN_DATA_TYPE_NOT_NULL NOT NULL,
       CHARACTER_MAXIMUM_LENGTH           INFORMATION_SCHEMA.CARDINAL_NUMBER,
       CHARACTER_OCTET_LENGTH             INFORMATION_SCHEMA.CARDINAL_NUMBER,
       COLLATION_CATALOG                  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_SCHEMA                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_NAME                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NUMERIC_PRECISION                  INFORMATION_SCHEMA.CARDINAL_NUMBER,
       NUMERIC_PRECISION_RADIX            INFORMATION_SCHEMA.CARDINAL_NUMBER,
       NUMERIC_SCALE                      INFORMATION_SCHEMA.CARDINAL_NUMBER,
       DATETIME_PRECISION                 INFORMATION_SCHEMA.CARDINAL_NUMBER,
       INTERVAL_CODE                      INFORMATION_SCHEMA.CHARACTER_DATA,
       INTERVAL_PRECISION                 INFORMATION_SCHEMA.CARDINAL_NUMBER,
       ABSTRACT_DATA_TYPE_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ABSTRACT_DATA_TYPE_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ABSTRACT_DATA_TYPE_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT TABLE_OR_DOMAIN_CHECK_COMBINATIONS
         CHECK ( DATA_TYPE IN ( 'CHARACTER', 'CHARACTER VARYING',
                                'BIT', 'BIT VARYING' )
               AND ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
                   COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                     IS NOT NULL
               AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX,
                     NUMERIC_SCALE, DATETIME_PRECISION,
                     ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                     ABSTRACT_DATA_TYPE_NAME ) IS NULL
               AND ( INTERVAL_CODE, INTERVAL_PRECISION )
                     IS NULL
         OR
               DATA_TYPE IN ( 'REAL', 'DOUBLE PRECISION', 'FLOAT' )
               AND ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
                   COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                       IS NULL
               AND NUMERIC_PRECISION IS NOT NULL
               AND NUMERIC_PRECISION_RADIX = 2
               AND NUMERIC_SCALE IS NULL
               AND DATETIME_PRECISION IS NULL
               AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                     ABSTRACT_DATA_TYPE_NAME ) IS NULL
               AND ( INTERVAL_CODE, INTERVAL_PRECISION )
                     IS NULL
         OR
               DATA_TYPE IN ( 'INTEGER', 'SMALLINT' )
               AND ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
                     COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
```

```
                IS NULL
        AND NUMERIC_PRECISION_RADIX IN ( 2, 10 ) )
        AND NUMERIC_PRECISION IS NOT NULL
        AND NUMERIC_SCALE = 0
        AND DATETIME_PRECISION IS NULL
        AND ( INTERVAL_CODE, INTERVAL_PRECISION ) IS NULL
OR
        DATA_TYPE IN ( 'NUMERIC', 'DECIMAL' )
        AND ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
              COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                      IS NULL
        AND NUMERIC_PRECISION_RADIX = 10
        AND ( NUMERIC_PRECISION, NUMERIC_SCALE ) IS NOT NULL
        AND DATETIME_PRECISION IS NULL
        AND ( INTERVAL_CODE, INTERVAL_PRECISION ) IS NULL
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
OR
        DATA_TYPE IN ( 'DATE', 'TIME', 'TIMESTAMP',
                       'TIME WITH TIME ZONE', 'TIMESTAMP WITH TIME ZONE' )
        AND ( CHARACTER_MAXIMUM_LENGTH,
            CHARACTER_OCTET_LENGTH,
            COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                IS NULL
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NOT NULL
        AND NUMERIC_SCALE IS NULL
        AND DATETIME_PRECISION IS NOT NULL
        AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                ABSTRACT_DATA_TYPE_NAME ) IS NULL
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
OR
        DATA_TYPE = 'INTERVAL'
        AND ( CHARACTER_MAXIMUM_LENGTH,
            CHARACTER_OCTET_LENGTH,
            COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                IS NULL
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NOT NULL
        AND NUMERIC_SCALE IS NULL
        AND DATETIME_PRECISION IS NOT NULL
        AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                ABSTRACT_DATA_TYPE_NAME ) IS NULL
        AND INTERVAL_CODE IN
                                ( 'YEAR', 'MONTH', 'DAY', 'HOUR',
                                  'MINUTE', 'SECOND', 'YEAR TO MONTH',
                                  'DAY TO HOUR', 'DAY TO MINUTE',
                                  'DAY TO SECOND', 'HOUR TO MINUTE',
                                  'HOUR TO SECOND', 'MINUTE TO SECOND' )
        AND INTERVAL_PRECISION
          IS NOT NULL
OR
        DATA_TYPE = 'BOOLEAN'
        AND ( CHARACTER_MAXIMUM_LENGTH,
            CHARACTER_OCTET_LENGTH,
            COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                IS NULL
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX ) IS NULL
        AND NUMERIC_SCALE IS NULL
        AND DATETIME_PRECISION IS NULL
        AND ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                                ABSTRACT_DATA_TYPE_NAME ) IS NULL
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
OR
```

```
        DATA_TYPE = 'USER_DEFINED'
        AND ( NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX )
              NUMERIC_SCALE, DATETIME_PRECISION,
            CHARACTER_OCTET_LENGTH,
              CHARACTER_MAXIMUM_LENGTH ) IS NULL
        AND ( INTERVAL_CODE, INTERVAL_PRECISION )
              IS NULL
  ),

CONSTRAINT DATA_TYPE_DESCRIPTOR_PRIMARY_KEY
  PRIMARY KEY ( TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                  TABLE_OR_DOMAIN_NAME, COLUMN_NAME ),

CONSTRAINT DATA_TYPE_CHECK_REFERENCES_COLLATION
    CHECK ( COLLATION_CATALOG
            <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
       OR
          ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ) IN
          ( SELECT COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME
            FROM COLLATIONS ) ),

CONSTRAINT DATA_TYPE_DESCRIPTOR_FOREIGN_KEY_SCHEMATA
  FOREIGN KEY ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA)
  REFERENCES SCHEMATA,

CONSTRAINT DATA_TYPE_DESCRIPTOR_CHECK_USED
  CHECK ( (TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
             TABLE_OR_DOMAIN_NAME, COLUMN_NAME)
    IN (
        SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
            FROM COLUMNS
        UNION
        SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, ''
            FROM DOMAINS )
)
```

Description

1) The values of TABLE_OR_DOMAIN_CATALOG and TABLE_OR_DOMAIN_SCHEMA are the catalog name and the unqualified schema name, respectively, of the schema that contains the object (domain or column) to which the data type descriptor belongs.

2) Case:

   a) If the length of COLUMN_NAME is 0, then the value of TABLE_OR_DOMAIN_NAME is the name of the domain to which the data type descriptor belongs.

   b) Otherwise, TABLE_OR_DOMAIN_NAME is the name of the table and COLUMN_NAME is the name of the column in that table to which the data type descriptor belongs.

3) The values of DATA_TYPE, CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH, COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, NUMERIC_PRECISION, NUMERIC_PRECISION_RADIX, NUMERIC_SCALE, DATETIME_PRECISION, ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA, and ABSTRACT_DATA_TYPE_NAME contain the data type of the domain or column being defined, the maximum length in characters or bits of the column if it is a character or bit type respectively, maximum length in octets of the column if it is a character type, the qualified name of the applicable collation if it is a character type, the precision and radix of the precision if it is a numeric type, the

scale if it is a numeric type, the fractional seconds precision if it is a datetime or interval type, and the qualified name of the abstract data type, if specified.

4) If DATA_TYPE is 'INTERVAL', then the values of INTERVAL_CODE are the value for <interval qualifier> (as specified in Table 4, "Codes used for <interval qualifier>s in Dynamic SQL"; see Part Z) for the data type being described; otherwise, INTERVAL_CODE is the general null value.

5) If DATA_TYPE is 'INTERVAL', then the values of INTERVAL_PRECISION are the interval leading field precision of the data type being described; otherwise, INTERVAL_PRECISION is the general null value.

6) The values of ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA, and ABSTRACT_DATA_TYPE_NAME are null if the column being described is not defined as an abstract data type. Otherwise, the values of ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA, and ABSTRACT_DATA_TYPE_NAME are the qualified name of the abstract data type used by the column being described.

18.4.8 DOMAINS base table

Function

The DOMAINS table has one row for each domain. It effectively contains a representation of the domain descriptors.

Definition

```
CREATE TABLE DOMAINS
       (
       DOMAIN_CATALOG        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_SCHEMA         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_DEFAULT        INFORMATION_SCHEMA.CHARACTER_DATA,
       NULL_CLASS_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NULL_CLASS_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NULL_CLASS_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT DOMAINS_PRIMARY_KEY
         PRIMARY KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ),

       CONSTRAINT DOMAINS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA ) REFERENCES SCHEMATA,

       CONSTRAINT DOMAINS_FOREIGN_KEY_NULL_CLASSES
         FOREIGN KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME )
           REFERENCES NULL_CLASSES,

       CONSTRAINT DOMAIN_CHECK_DATA_TYPE
         CHECK ( DOMAIN_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
             OR
                ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME, '' ) IN
                ( SELECT TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                         TABLE_OR_DOMAIN_NAME, COLUMN_NAME
                  FROM DATA_TYPE_DESCRIPTOR ) ),
       )
```

Description

1) The values of DOMAIN_CATALOG and DOMAIN_SCHEMA are the catalog name and unqualified schema name, respectively, of the schema in which the domain is defined.

2) The value of DOMAIN_NAME is the name of the domain.

3) The value of DOMAIN_DEFAULT is null if the domain being described has no explicit default value. If the character representation of the default value cannot be represented without truncation, then the value of COLUMN_DEFAULT is “TRUNCATED”. Otherwise, the value of DOMAIN_DEFAULT is a character representation of the default value for the domain that obeys the rules specified for <default option> in Subclause 11.9, "<default clause>".

   Note: “TRUNCATED” is different from other values like USER or CURRENT_TIMESTAMP in that it is not an SQL <key word> and does not correspond to a defined value in SQL.

4) If the null class of the domain is the general null class, then the values of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME are null; otherwise, the values of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME are the <catalog name>, <unqualified schema name>, and <qualified identifier>, respectively, of the <null class name> of the null class of the domain.

18.4.9 DOMAIN_CONSTRAINTS base table

Function

The DOMAIN_CONSTRAINTS table has one row for each domain constraint associated with a domain. It effectively contains a representation of the domain constraint descriptors.

Definition

```
CREATE TABLE DOMAIN_CONSTRAINTS
       (
       CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_CATALOG        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT DOMAIN_CATALOG_NOT_NULL NOT NULL,
       DOMAIN_SCHEMA         INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT DOMAIN_SCHEMA_NOT_NULL NOT NULL,
       DOMAIN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT DOMAIN_NAME_NOT_NULL NOT NULL,
       IS_DEFERRABLE         INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT DOMAIN_CONSTRAINTS_DEFERRABLE_NOT_NULL NOT NULL,
       INITIALLY_DEFERRED    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT DOMAIN_CONSTRAINTS_INITIALLY_DEFERRED_NOT_NULL NOT NULL,
       CONSTRAINT DOMAIN_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),
       CONSTRAINT DOMAIN_CONSTRAINTS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT DOMAIN_CONSTRAINTS_FOREIGN_KEY_CHECK_CONSTRAINTS
         FOREIGN KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA, CONSTRAINT_NAME )
           REFERENCES CHECK_CONSTRAINTS,

       CONSTRAINT DOMAIN_CONSTRAINTS_FOREIGN_KEY_DOMAINS
         FOREIGN KEY ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME )
           REFERENCES DOMAINS,

       CONSTRAINT DOMAIN_CONSTRAINTS_CHECK_DEFERRABLE
         CHECK ( ( IS_DEFERRABLE, INITIALLY_DEFERRED ) IN
                 ( VALUES ( 'NO',  'NO' ),
                          ( 'YES', 'NO' ),
                          ( 'YES', 'YES' ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG and CONSTRAINT_SCHEMA are the catalog name and unqualified schema name of the schema in which the domain constraint is defined.

2) The value of CONSTRAINT_NAME is the name of the domain constraint.

3) The values of DOMAIN_CATALOG, DOMAIN_SCHEMA and DOMAIN_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the domain in which the domain constraint is defined.

4) The values of IS_DEFERRABLE have the following meanings:

YES	The domain constraint is deferrable.
NO	The domain constraint is not deferrable.

5) The values of INITIALLY_DEFERRED have the following meanings:

YES	The domain constraint is initially deferred.
NO	The domain constraint is initially immediate.

18.4.10 TABLES base table

Function

The TABLES table contains one row for each table including views. It effectively contains a representation of the table descriptors.

Definition

```
CREATE TABLE TABLES
       (
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_TYPE             INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_TYPE_NOT_NULL NOT NULL,
         CONSTRAINT TABLE_TYPE_CHECK CHECK ( TABLE_TYPE IN
                 ( 'BASE TABLE', 'VIEW', 'GLOBAL TEMPORARY', 'LOCAL TEMPORARY' ) ),
         CONSTRAINT CHECK_TABLE_IN_COLUMNS
           CHECK ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
                 ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                     FROM COLUMNS ) ),

       CONSTRAINT TABLES_PRIMARY_KEY
         PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

       CONSTRAINT TABLES_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT TABLES_CHECK_NOT_VIEW CHECK ( NOT EXISTS
         ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
             FROM TABLES
             WHERE TABLE_TYPE = 'VIEW'
           EXCEPT
           SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
             FROM VIEWS ) )
       )
```

Description

1) The values of TABLE_CATALOG and TABLE_SCHEMA are the catalog name and unqualified schema name, respectively, of the schema in which the table is defined.

2) The value of TABLE_NAME is the name of the table.

3) The values of TABLE_TYPE have the following meanings:

BASE TABLE	The table being described is a persistent base table.
VIEW	The table being described is a viewed table.
GLOBAL TEMPORARY	The table being described is a global temporary table.
LOCAL TEMPORARY	The table being described is a created local temporary table.

18.4.11 VIEWS base table

Function

The VIEWS table contains one row for each row in the TABLES table with a TABLE_TYPE of 'VIEW'. Each row describes the query expression that defines a view. The table effectively contains a representation of the view descriptors.

Definition

```
CREATE TABLE VIEWS
       (
       TABLE_CATALOG   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_DEFINITION INFORMATION_SCHEMA.CHARACTER_DATA,
       CHECK_OPTION    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT CHECK_OPTION_NOT_NULL NOT NULL
         CONSTRAINT CHECK_OPTION_CHECK
           CHECK ( CHECK_OPTION IN ( 'CASCADED', 'LOCAL', 'NONE' ) ),
       IS_UPDATABLE       INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT IS_UPDATABLE_NOT_NULL NOT NULL
         CONSTRAINT IS_UPDATABLE_CHECK
           CHECK ( IS_UPDATABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT VIEWS_PRIMARY_KEY
         PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

       CONSTRAINT VIEWS_IN_TABLES_CHECK
         CHECK ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
         ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
             FROM TABLES
             WHERE TABLE_TYPE = 'VIEW' ) ),

       CONSTRAINT VIEWS_IS_UPDATABLE_CHECK_OPTION_CHECK
         CHECK ( ( IS_UPDATABLE, CHECK_OPTION ) NOT IN
           ( VALUES ( 'NO', 'CASCADED' ), ( 'NO', 'LOCAL' ) ) )
       )
```

Description

1) The values of TABLE_CATALOG and TABLE_SCHEMA are the catalog name and unqualified schema name, respectively, of the schema in which the viewed table is defined.

2) The value of TABLE_NAME is the name of the viewed table.

3) Case:

   a) If the character representation of the <query expression> contained in the <view definition> that defined the view being described can be represented without truncation, then the value of VIEW_DEFINITION is that character representation.

   b) Otherwise, the value of VIEW_DEFINITION is the null value.

Note: Any implicit column references that were contained in the <query expression> associated with the <view definition> are replaced by explicit column references in VIEW_DEFINITION.

4) The values of CHECK_OPTION have the following meanings:

CASCADED	The <view definition> contains WITH CASCADED CHECK OPTION.
LOCAL	The <view definition> contains WITH LOCAL CHECK OPTION.
NONE	The <view definition> does not contain WITH CHECK OPTION.

5) The values of IS_UPDATABLE have the following meanings:

YES	The <view definition> simply contains a <query expression> that is inherently updatable.
NO	The <view definition> simply contains a <query expression> that is not inherently updatable.

18.4.12 COLUMNS base table

Function

The COLUMNS table has one row for each non-inherited column. It effectively contains a representation of the column descriptors.

Definition

```
CREATE TABLE COLUMNS
       (
       TABLE_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ORDINAL_POSITION      INFORMATION_SCHEMA.CARDINAL_NUMBER
         CONSTRAINT COLUMN_POSITION_NOT_NULL NOT NULL,
       DOMAIN_CATALOG        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_SCHEMA         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       DOMAIN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_DEFAULT        INFORMATION_SCHEMA.CHARACTER_DATA,
       IS_NULLABLE           INFORMATION_SCHEMA.CHARACTER_DATA
       CONSTRAINT IS_NULLABLE_NOT_NULL NOT NULL
       CONSTRAINT IS_NULLABLE_CHECK
         CHECK ( IS_NULLABLE IN ('YES', 'NO' ),
       NULL_CLASS_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NULL_CLASS_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NULL_CLASS_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT COLUMNS_PRIMARY_KEY
         PRIMARY KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ),

       CONSTRAINT COLUMNS_UNIQUE
         UNIQUE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, ORDINAL_POSITION ),

       CONSTRAINT COLUMNS_FOREIGN_KEY_TABLES
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
           REFERENCES TABLES,

       CONSTRAINT COLUMNS_FOREIGN_KEY_NULL_CLASSES
         FOREIGN KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME )
           REFERENCES NULL_CLASSES,

       CONSTRAINT COLUMNS_CHECK_REFERENCES_DOMAIN
         CHECK ( DOMAIN_CATALOG
                 <> ANY ( SELECT  CATALOG_NAME FROM SCHEMATA )
             OR
                ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IN
                ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME
                  FROM DOMAINS ) ),

       CONSTRAINT COLUMN_CHECK_DATA_TYPE
         CHECK ( DOMAIN_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
             OR
            ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IS NOT NULL
                AND
              ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) NOT IN
              ( SELECT TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                         TABLE_OR_DOMAIN_NAME, COLUMN_NAME
                  FROM DATA_TYPE_DESCRIPTOR )
```

```
          AND
            ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME ) IS NULL
        OR
          ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME ) IS NULL
          AND
          ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) IN
          ( SELECT TABLE_OR_DOMAIN_CATALOG, TABLE_OR_DOMAIN_SCHEMA,
                   TABLE_OR_DOMAIN_NAME, COLUMN_NAME
            FROM DATA_TYPE_DESCRIPTOR )
          AND
            ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME ) IS NOT NULL
      ) )
  )
```

Description

1) Case:

   a) If a column is described by a column descriptor included in a table descriptor, then the table descriptor and the column descriptor are associated with that column.

   b) If a column is described by a column descriptor included in a view descriptor, then the view descriptor and the corresponding column descriptor of the table of the <query expression> are associated with that column.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the table containing the column being described.

3) The value of COLUMN_NAME is the name of the column being described.

4) The values of DOMAIN_CATALOG, DOMAIN_SCHEMA, and DOMAIN_NAME are null if the column being described is not defined using a <domain name>. Otherwise, the values of DOMAIN_CATALOG, DOMAIN_SCHEMA, and DOMAIN_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the domain used by the column being described.

5) If the null class of the domain is the general null class, then the values of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME are null; otherwise, the values of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME are the <catalog name>, <unqualified schema name>, and <qualified identifier>, respectively, of the <null class name> of the null class of the column.

6) The value of ORDINAL_POSITION is the ordinal position of the column in the table.

7) The value of COLUMN_DEFAULT is null if the column being described has no explicit default value or if its default value comes only from a domain. If the character representation of the default value cannot be represented without truncation, then the value of COLUMN_DEFAULT is “TRUNCATED”. Otherwise, the value of COLUMN_DEFAULT is a character representation of the default value for the column that obeys the rules specified for <default option> in Subclause 11.9, "<default clause>".

   Note: “TRUNCATED” is different from other values like USER or CURRENT_TIMESTAMP in that it is not an SQL <key word> and does not correspond to a defined value in SQL.

8) The values of IS_NULLABLE have the following meanings:

YES	The columns is possibly nullable.
NO	The column is known not nullable.

18.4.13 VIEW_TABLE_USAGE base table

Function

The VIEW_TABLE_USAGE table has one row for each table identified by a <table name> simply contained in a <table reference> that is contained in the <query expression> of a view.

Definition

```
CREATE TABLE VIEW_TABLE_USAGE
       (
       VIEW_CATALOG           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_SCHEMA            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT VIEW_TABLE_USAGE_PRIMARY_KEY
       PRIMARY KEY ( VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME,
                     TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

       CONSTRAINT VIEW_TABLE_USAGE_CHECK_REFERENCES_TABLES
           CHECK ( TABLE_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                  ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
                  ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                    FROM TABLES ) ),

       CONSTRAINT VIEW_TABLE_USAGE_FOREIGN_KEY_VIEWS
       FOREIGN KEY ( VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME )
         REFERENCES VIEWS
       )
```

Description

1) The values of VIEW_CATALOG, VIEW_SCHEMA, and VIEW_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the view being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of a table identified by a <table name> simply contained in a <table reference> that is contained in the <query expression> of the view being described.

18.4.14 VIEW_COLUMN_USAGE base table

Function

The VIEW_COLUMN_USAGE table has one row for each column of a table identified by a <table name> simply contained in a <table reference> that is contained in the <query expression> of the view that is explicitly or implicitly referenced in the <query expression> of the view being described.

Definition

```
CREATE TABLE VIEW_COLUMN_USAGE
       (
       VIEW_CATALOG           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_SCHEMA            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       VIEW_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT VIEW_COLUMN_USAGE_PRIMARY_KEY
       PRIMARY KEY ( VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME,
                     TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ),

       CONSTRAINT VIEW_COLUMN_USAGE_CHECK_REFERENCES_COLUMNS
           CHECK ( TABLE_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                  ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) IN
                  ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
                    FROM COLUMNS ) ),

       CONSTRAINT VIEW_COLUMN_USAGE_FOREIGN_KEY_VIEWS
       FOREIGN KEY ( VIEW_CATALOG, VIEW_SCHEMA, VIEW_NAME )
         REFERENCES VIEWS
       )
```

Description

1) The values of VIEW_CATALOG, VIEW_SCHEMA, and VIEW_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the view being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalog name, unqualified schema name, qualified identifier, and column name, respectively, of a column of a table identified by a <table name> simply contained in a <table reference> that is contained in that is explicitly or implicitly referenced in the <query expression> of the view being described.

18.4.15 TABLE_CONSTRAINTS base table

Function

The TABLE_CONSTRAINTS table has one row for each table constraint associated with a table. It effectively contains a representation of the table constraint descriptors.

Definition

```
CREATE TABLE TABLE_CONSTRAINTS
       (
       CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_TYPE       INFORMATION_SCHEMA.CHARACTER_DATA
       CONSTRAINT CONSTRAINT_TYPE_NOT_NULL NOT NULL
         CONSTRAINT CONSTRAINT_TYPE_CHECK

           CHECK ( CONSTRAINT_TYPE IN
                   ( 'UNIQUE',
                     'PRIMARY KEY',
                     'FOREIGN KEY',
                     'CHECK' ) ),

       TABLE_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TABLE_CONSTRAINTS_TABLE_CATALOG_NOT_NULL NOT NULL,
       TABLE_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TABLE_CONSTRAINTS_TABLE_SCHEMA_NOT_NULL NOT NULL,
       TABLE_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TABLE_CONSTRAINTS_TABLE_NAME_NOT_NULL NOT NULL,
       IS_DEFERRABLE INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_CONSTRAINTS_IS_DEFERRABLE_NOT_NULL NOT NULL,
       INITIALLY_DEFERRED    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_CONSTRAINTS_INITIALLY_DEFERRED_NOT_NULL

       CONSTRAINT TABLE_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),
```

- 1 constraint deleted.

```
       CONSTRAINT TABLE_CONSTRAINTS_DEFERRED_CHECK
         CHECK ( ( IS_DEFERRABLE, INITIALLY_DEFERRED ) IN
                 ( VALUES ( 'NO',  'NO' ),
                          ( 'YES', 'NO' ),
                          ( 'YES', 'YES' ) ) ),

       CONSTRAINT TABLE_CONSTRAINTS_CHECK_VIEWS
         CHECK ( TABLE_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
             OR
            ( ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
              ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
               FROM TABLES
               WHERE TABLE_TYPE <> 'VIEW' ) ) ),

       CONSTRAINT TABLE_CONSTRAINTS_UNIQUE_CHECK
         CHECK ( 1 =
            ( SELECT COUNT (*)
              FROM ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                     FROM TABLE_CONSTRAINTS
                     WHERE CONSTRAINT_TYPE IN
```

```
                        ( 'UNIQUE', 'PRIMARY KEY' )
        UNION ALL
        SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
        FROM REFERENTIAL_CONSTRAINTS
        UNION ALL
        SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
        FROM CHECK_CONSTRAINTS ) AS X
        WHERE ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
            = ( X.CONSTRAINT_CATALOG, X.CONSTRAINT_SCHEMA, X.CONSTRAINT_NAME )
    ) ),

CONSTRAINT UNIQUE_TABLE_PRIMARY_KEY_CHECK
  CHECK ( UNIQUE ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                   FROM TABLE_CONSTRAINTS
                   WHERE CONSTRAINT_TYPE = 'PRIMARY KEY' ) )
)
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described. If the <table constraint definition> or <add table constraint definition> that defined the constraint did not specify a <constraint name>, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are implementation-defined.

2) The values of CONSTRAINT_TYPE have the following meanings:

FOREIGN KEY	The constraint being described is a foreign key constraint.
UNIQUE	The constraint being described is a unique constraint.
PRIMARY KEY	The constraint being described is a primary key constraint.
CHECK	The constraint being described is a check constraint.

3) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, the unqualified schema name, and the qualified identifier of the name of the table to which the table constraint being described applies.

4) The values of IS_DEFERRABLE have the following meanings:

YES	The table constraint is deferrable.
NO	The table constraint is not deferrable.

5) The values of INITIALLY_DEFERRED have the following meanings:

YES	The table constraint is initially deferred.
NO	The table constraint is initially immediate.

18.4.16 KEY_COLUMN_USAGE base table

Function

The KEY_COLUMN_USAGE table has one or more rows for each row in the TABLE_CONSTRAINTS table that has a CONSTRAINT_TYPE of “UNIQUE”, “PRIMARY KEY”, or “FOREIGN KEY”. The rows list the columns that constitute each unique constraint, and the referencing columns in each foreign key constraint.

Definition

```
CREATE TABLE KEY_COLUMN_USAGE
       (
       CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT KEY_COLUMN_TABLE_CATALOG_NOT_NULL NOT NULL,
       TABLE_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT KEY_COLUMN_TABLE_SCHEMA_NOT_NULL NOT NULL,
       TABLE_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT KEY_COLUMN_TABLE_NAME_NOT_NULL NOT NULL,
       COLUMN_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ORDINAL_POSITION      INFORMATION_SCHEMA.CARDINAL_NUMBER
         CONSTRAINT KEY_COLUMN_ORDINAL_POSITION_NOT_NULL NOT NULL,

       CONSTRAINT KEY_COLUMN_USAGE_PRIMARY_KEY
       PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
                     COLUMN_NAME ),

       CONSTRAINT KEY_COLUMN_USAGE_UNIQUE
       UNIQUE ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA,
                CONSTRAINT_NAME, ORDINAL_POSITION ),

       CONSTRAINT KEY_COLUMN_USAGE_FOREIGN_KEY_COLUMNS
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME )
           REFERENCES COLUMNS,

       CONSTRAINT KEY_COLUMN_CONSTRAINT_TYPE_CHECK
       CHECK ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
               IN ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                      FROM TABLE_CONSTRAINTS
                      WHERE CONSTRAINT_TYPE IN
                          ( 'UNIQUE', 'PRIMARY KEY', 'FOREIGN KEY' ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalog name, unqualified schema name, qualified identifier of the table name, and the column name of the column that participates in the unique, primary key, or foreign key constraint being described.

3) The value of ORDINAL_POSITION is the ordinal position of the specific column in the constraint being described. If the constraint described is a key of cardinality 1, then the value of ORDINAL_POSITION is always 1. If the constraint being described is a foreign key constraint, then ORDINAL_POSITION also identifies the position within the uniqueness constraint of the column that this column references.

18.4.17 REFERENTIAL_CONSTRAINTS base table

Function

The REFERENTIAL_CONSTRAINTS table has one row for each row in the TABLE_CONSTRAINTS table that has a CONSTRAINT_TYPE value of “FOREIGN KEY”.

Definition

```
CREATE TABLE REFERENTIAL_CONSTRAINTS
       (
       CONSTRAINT_CATALOG           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME              INFORMATION_SCHEMA.SQL_IDENTIFIER,

       UNIQUE_CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT UNIQUE_CONSTRAINT_CATALOG_NOT_NULL NOT NULL,
       UNIQUE_CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT UNIQUE_CONSTRAINT_SCHEMA_NOT_NULL NOT NULL,
       UNIQUE_CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT UNIQUE_CONSTRAINT_NAME_NOT_NULL NOT NULL,
       MATCH_OPTION                 INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT REFERENTIAL_MATCH_OPTION_NOT_NULL NOT NULL
         CONSTRAINT REFERENTIAL_MATCH_OPTION_CHECK
           CHECK ( MATCH_OPTION IN ( 'NONE', 'PARTIAL', 'FULL' ) ),
       UPDATE_RULE                  INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT REFERENTIAL_UPDATE_RULE_NOT_NULL NOT NULL
         CONSTRAINT REFERENTIAL_UPDATE_RULE_CHECK
           CHECK ( UPDATE_RULE IN
                   ('CASCADE', 'SET NULL', 'SET DEFAULT', 'RESTRICT', 'NO ACTION' ) ),
       DELETE_RULE                  INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT REFERENTIAL_DELETE_RULE_NOT_NULL NOT NULL
         CONSTRAINT REFERENTIAL_DELETE_RULE_CHECK
           CHECK ( DELETE_RULE IN
                   ('CASCADE', 'SET NULL', 'SET DEFAULT', 'RESTRICT', 'NO ACTION' ) ),

       CONSTRAINT REFERENTIAL_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

       CONSTRAINT REFERENTIAL_CONSTRAINTS_CONSTRAINT_TYPE_CHECK
         CHECK ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
               IN ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                      FROM TABLE_CONSTRAINTS
                      WHERE CONSTRAINT_TYPE = 'FOREIGN KEY' ) ),

       CONSTRAINT UNIQUE_CONSTRAINT_CHECK_REFERENCES_UNIQUE_CONSTRAINT
         CHECK ( UNIQUE_CONSTRAINT_CATALOG <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
               ( ( UNIQUE_CONSTRAINT_CATALOG, UNIQUE_CONSTRAINT_SCHEMA,
                   UNIQUE_CONSTRAINT_NAME ) IN
                     ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                       FROM TABLE_CONSTRAINTS
                       WHERE CONSTRAINT_TYPE
                               IN ( 'UNIQUE', 'PRIMARY KEY' ) ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) The values of UNIQUE_CONSTRAINT_CATALOG, UNIQUE_CONSTRAINT_SCHEMA, and UNIQUE_CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the unique or primary key constraint applied to the referenced column list being described.

3) The values of MATCH_OPTION have the following meanings:

NONE	No <match type> was specified.
PARTIAL	A <match type> of PARTIAL was specified.
FULL	A <match type> of FULL was specified.

4) The values of UPDATE_RULE have the following meanings for a referential constraint that has an <update rule>:

NO ACTION	A <referential action> of NO ACTION was specified.
RESTRICT	A <referential action> of RESTRICT was specified.
CASCADE	A <referential action> of CASCADE was specified.
SET NULL	A <referential action> of SET NULL was specified.
SET DEFAULT	A <referential action> of SET DEFAULT was specified.

5) The values of DELETE_RULE have the following meanings for a referential constraint that has a <delete rule>:

NO ACTION	A <referential action> of NO ACTION was specified.
RESTRICT	A <referential action> of RESTRICT was specified.
CASCADE	A <referential action> of CASCADE was specified.
SET NULL	A <referential action> of SET NULL was specified.
SET DEFAULT	A <referential action> of SET DEFAULT was specified.

18.4.18 CHECK_CONSTRAINTS base table

Function

The CHECK_CONSTRAINTS table has one row for each domain constraint, table check constraint, and assertion.

Definition

```
CREATE TABLE CHECK_CONSTRAINTS
       (
       CONSTRAINT_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CHECK_CLAUSE       INFORMATION_SCHEMA.CHARACTER_DATA,

       CONSTRAINT CHECK_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

       CONSTRAINT CHECK_CONSTRAINTS_SOURCE_CHECK
         CHECK ( ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
                     IN
                       ( SELECT * FROM (
                           SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME

                           FROM ASSERTIONS
                           UNION
                           SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME

                           FROM TABLE_CONSTRAINTS
                           UNION
                           SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME

                           FROM DOMAIN_CONSTRAINTS ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) Case:

   a) If the character representation of the <search condition> contained in the <check constraint definition>, <domain constraint definition>, or <assertion definition> that defined the check constraint being described can be represented without truncation, then the value of CHECK_CLAUSE is that character representation.

   b) Otherwise, the value of CHECK_CLAUSE is the null value.

   Note: Any implicit column references that were contained in the <search condition> associated with a <check constraint definition> or an <assertion definition> are replaced by explicit column references in CHECK_CONSTRAINTS.

18.4.19 CHECK_TABLE_USAGE base table

Function

The CHECK_TABLE_USAGE table has one row for each table identified by a <table name> simply contained in a <table reference> contained in the <search condition> of a check constraint, domain constraint, or assertion.

Definition

```
CREATE TABLE CHECK_TABLE_USAGE
       (
       CONSTRAINT_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT CHECK_TABLE_USAGE_PRIMARY_KEY
       PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
                     TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ),

       CONSTRAINT CHECK_TABLE_USAGE_FOREIGN_KEY_CHECK_CONSTRAINTS
       FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
         REFERENCES CHECK_CONSTRAINTS,

       CONSTRAINT CHECK_TABLE_USAGE_CHECK_REFERENCES_TABLES
           CHECK ( TABLE_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
              OR
                 ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME ) IN
                 ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                   FROM TABLES ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of a table identified by a <table name> simply contained in a <table reference> contained in the <search condition> of the constraint being described.

18.4.20 CHECK_COLUMN_USAGE base table

Function

The CHECK_COLUMN_USAGE table has one row for each column identified by a <column reference> contained in the <search condition> of a check constraint, domain constraint, or assertion.

Definition

```
CREATE TABLE CHECK_COLUMN_USAGE
       (
       CONSTRAINT_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT CHECK_COLUMN_USAGE_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME,
                       TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ),

       CONSTRAINT CHECK_COLUMN_USAGE_FOREIGN_KEY_CHECK_CONSTRAINTS
         FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
           REFERENCES CHECK_CONSTRAINTS,

       CONSTRAINT CHECK_COLUMN_USAGE_CHECK_REFERENCES_COLUMNS
         CHECK ( TABLE_CATALOG
                   <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
                 OR
                 ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME ) IN
                   ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
                     FROM COLUMNS ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the constraint being described.

2) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalog name, unqualified schema name, qualified identifier, and column name, respectively, of a column identified by a <column reference> explicitly or implicitly contained in the <search condition> of the constraint being described.

18.4.21 ABSTRACT_DATA_TYPE_PRIVILEGES base table

Function

The ABSTRACT_DATA_TYPE_PRIVILEGES table has one row for each abstract data type privilege descriptor. It effectively contains a representation of the privilege descriptors.

Definition

```
CREATE TABLE ABSTRACT_DATA_TYPE_PRIVILEGES
       (
         GRANTOR                        INFORMATION_SCHEMA.SQL_IDENTIFIER,
         GRANTEE                        INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ABSTRACT_DATA_TYPE_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,

• 1 column deleted.
         PRIVILEGE_TYPE                 INFORMATION_SCHEMA.CHARACTER_DATA
           CONSTRAINT PRIVILEGE_TYPE_CHECK
             CHECK ( PRIVILEGE_TYPE = 'DATA TYPE USAGE' ),
         IS_GRANTABLE                   INFORMATION_SCHEMA.CHARACTER_DATA
           ABSTRACT_DATA_TYPE_CONSTRAINT IS_GRANTABLE_NOT_NULL
             NOT NULL
           ABSTRACT_DATA_TYPE_CONSTRAINT IS_GRANTABLE_CHECK
             CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),
         CONSTRAINT ABSTRACT_DATA_TYPE_PRIMARY_KEY
           PRIMARY KEY(GRANTOR, GRANTEE,
                       ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                       ABSTRACT_DATA_TYPE_NAME, ABSTRACT_DATA_TYPE_LANGUAGE,
                       PRIVILEGE_TYPE ),
         CONSTRAINT ABSTRACT_DATA_TYPE_PRIVILEGES_FOREIGN_KEY_ABSTRACT_DATA_TYPE
           FOREIGN KEY ( ABSTRACT_DATA_TYPE_CATALOG, ABSTRACT_DATA_TYPE_SCHEMA,
                         ABSTRACT_DATA_TYPE_NAME )
             REFERENCES ABSTRACT_DATA_TYPES,
         CONSTRAINT ABSTRACT_DATA_TYPE_PRIVILEGES_FOREIGN_KEY_GRANTOR
           FOREIGN KEY ( GRANTOR )
             REFERENCES USERS,
         CONSTRAINT ABSTRACT_DATA_TYPE_FOREIGN_KEY_GRANTEE
           FOREIGN KEY ( GRANTEE )
             REFERENCES USERS
       )
```

Description

1) A row is inserted into this table when a <grant statement> is executed, unless the necessary row already exists, in which case the existing row may be modified to change the IS_GRANTABLE column. One or more rows are deleted from this table when a <revoke statement> is executed.

2) The value of GRANTOR is the <authorization identifier> of the user who granted access privileges on the DATA TYPE USAGE privilege being described to the user identified by the value of GRANTEE.

3) The value of GRANTEE is the <authorization identifier>, possibly "PUBLIC", of some <grantee> contained in a <grant statement>.

4) The value of PRIVILEGE_TYPE has the following meaning:

DATA TYPE USAGE	The user has DATA TYPE USAGE privilege on this abstract data type.

5) The values of IS_GRANTABLE have the following meanings:

YES	The privilege being described was granted WITH GRANT OPTION and is thus grantable
NO	The privilege being described was not granted WITH GRANT OPTION and is thus not grantable

18.4.22 ASSERTIONS base table

Function

The ASSERTIONS table has one row for each assertion. It effectively contains a representation of the assertion descriptors.

Definition

```
CREATE TABLE ASSERTIONS
       (
       CONSTRAINT_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       IS_DEFERRABLE         INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT ASSERTIONS_IS_DEFERRABLE_NOT_NULL NOT NULL
       INITIALLY_DEFERRED    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT ASSERTIONS_INITIALLY_DEFERRED_NOT_NULL NOT NULL
       CHECK_TIME            INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT ASSERTIONS_CHECK_TIME_CHECK
           CHECK ( CHECK_TIME IN ('IMMEDIATE', 'DEFERRED' ) ),

       CONSTRAINT ASSERTIONS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

       CONSTRAINT ASSERTIONS_FOREIGN_KEY_CHECK_CONSTRAINTS
         FOREIGN KEY (CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
           REFERENCES CHECK_CONSTRAINTS,

       CONSTRAINT ASSERTIONS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT ASSERTIONS_DEFERRED_CHECK
         CHECK ( ( IS_DEFERRABLE, INITIALLY_DEFERRED ) IN
            VALUES ( ( 'NO',  'NO' ),
                     ( 'YES', 'NO' ),
                     ( 'YES', 'YES' ) ) )
       )
```

Description

1) The values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the assertion being described.

2) The values of IS_DEFERRABLE have the following meanings:

   YES The assertion is deferrable.

   NO The assertion is not deferrable.

3) The values of INITIALLY_DEFERRED have the following meanings:

   YES The assertion is initially deferred.

   NO The assertion is initially immediate.

18.4.23 ROLE_AUTHORIZATION_DESCRIPTORS base table

Function

Contains a representation of the role authorization descriptors.

Definition

```
CREATE TABLE ROLE_AUTHORIZATION_DESCRIPTORS
       (
       ROLE_NAME                        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE                          INFORMATION_SCHEMA.SQL_IDENTIFIER
         CHECK ( GRANTEE IN ( SELECT ROLE_NAME FROM ROLES )
              OR GRANTEE IN ( SELECT USER_NAME FROM USERS ) ),
       IS_GRANTABLE                     INFORMATION_SCHEMA.CHARACTER_DATA
         CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT ROLE_AUTHORIZATION_DESCRIPTORS_PRIMARY_KEY
         PRIMARY KEY ( ROLE_NAME, GRANTEE ),
       CONSTRAINT ROLE_AUTHORIZATION_DESCRIPTORS_FOREIGN_KEY_ROLES
         FOREIGN KEY ( ROLE_NAME )
           REFERENCES ROLES,
       CONSTRAINT ROLE_AUTHORIZATION_DESCRIPTORS_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTEE )
           REFERENCES USERS
       )
```

Description

1) A row is (or rows are) inserted into this table whenever a <grant role statement> or <role definition> is executed unless the necessary row already exists, in which case the existing row may be modified to change the IS_GRANTABLE column. A row is (or rows are) deleted from this table whenever a <revoke role statement> or <drop role> is executed.

2) The value of ROLE_NAME is the <role name> of some <role granted> by the <grant role statement> or the <role name> of a <role definition>.

3) The value of GRANTEE is an <authorization identifier>, possibly PUBLIC, or <role name> specified as a <grantee> contained in a <grant role statement>, or the <authorization identifier> of the current SQL-session when the <role definition> is executed.

4) The values of IS_GRANTABLE have the following meanings:

YES	The described role is grantable.
NO	The described role is not grantable.

   A role is grantable if the WITH
   ANSI GRANT
   ISO ADMIN
   OPTION is specified in the <grant role statement> or a <role definition> is executed.

18.4.24 NULL_CLASSES base table

Function

The NULL_CLASSES table has one row for each null class.

Definition

```
CREATE TABLE NULL_CLASSES
       (
         NULL_CLASS_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_CLASS_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_CLASS_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONSTRAINT NULL_CLASSES_PRIMARY_KEY
           PRIMARY KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME ),
         CONSTRAINT NULL_CLASSES_FOREIGN_KEY_SCHEMATA
           FOREIGN KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA )
             REFERENCES SCHEMATA
       )
```

Description

1) A row is inserted in this table whenever a <null class definition> is executed; a row is deleted whenever a <drop null class statement> is executed.

2) The value of NULL_CLASS_CATALOG and NULL_CLASS_SCHEMA is the name of the schema in which the null class is defined.

18.4.25 NULL_STATES base table

Function

The NULL_STATES table has one row for each null state.

Definition

```
CREATE TABLE NULL_STATES
       (
         NULL_CLASS_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_CLASS_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_CLASS_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
         NULL_STATE_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
         ORDINAL_POSITION       INFORMATION_SCHEMA.CARDINAL_NUMBER,
         CONSTRAINT NULL_STATES_PRIMARY_KEY
           PRIMARY KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME,
                         NULL_STATE_NAME ),
         CONSTRAINT NULL_CLASSES_FOREIGN_KEY_NULL_CLASSES
           FOREIGN KEY ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME )
             REFERENCES NULL_CLASSES,
         CONSTRAINT NULL_CLASSES_UNIQUE
           UNIQUE ( NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA,
                    NULL_CLASS_NAME, ORDINAL_POSITION )
       )
```

Description

1) A row is inserted in this table for each null state in a null class whenever a <null class definition> is executed. Whenever a <drop null class statement> is executed, the rows in this table that correspond to the null states of that null class are deleted.

2) The value of NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, and NULL_CLASS_NAME is the name of the null class that contains the null state.

3) NULL_STATE_NAME is the name of the null state.

   Note: The fact that a null state in one null class may have the same name as some null state in some other null class is of no significance.

4) ORDINAL_POSITION is the position of the null state within the null class.

18.4.26 TABLE_PRIVILEGES base table

Function

The TABLE_PRIVILEGES table has one row for each table privilege descriptor. It effectively contains a representation of the table privilege descriptors.

Definition

```
CREATE TABLE TABLE_PRIVILEGES
       (
       GRANTOR                INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE                INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       PRIVILEGE_TYPE              INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_PRIVILEGE_TYPE_CHECK
           CHECK ( PRIVILEGE_TYPE IN
                    ( 'SELECT', 'INSERT', 'DELETE', 'UPDATE',
                      'TRIGGER', 'REFERENCES' ) ),
       IS_GRANTABLE           INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_PRIVILEGE_GRANTABLE_NOT_NULL NOT<REFERE
         CONSTRAINT TABLE_PRIVILEGE_GRANTABLE_CHECK
           CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT TABLE_PRIVILEGE_PRIMARY_KEY
         PRIMARY KEY ( GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                       PRIVILEGE_TYPE ),

       CONSTRAINT TABLE_PRIVILEGE_FOREIGN_KEY_TABLES
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
           REFERENCES TABLES,
       CONSTRAINT TABLE_PRIVILEGE_GRANTOR_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,


       CONSTRAINT TABLE_PRIVILEGE_GRANTEE_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTEE )
           REFERENCES USERS
       )
```

Description

1) The value of GRANTOR is the <authorization identifier> of the user who granted table privileges, on the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME, to the user identified by the value of GRANTEE for the table privilege being described.

2) The value of GRANTEE is the <authorization identifier> of some user, or “PUBLIC” to indicate all users, to whom the table privilege being described is granted.

3) The values of TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the table on which the privilege being described has been granted.

4) The values of PRIVILEGE_TYPE have the following meanings:

SELECT	The user has SELECT privileges on the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.
DELETE	The user has DELETE privileges on the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.
INSERT	The user will automatically be granted INSERT privileges on any columns that may be added to the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME in the future.
UPDATE	The user will automatically be granted UPDATE privileges on any columns that may be added to the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME in the future.
REFERENCES	The user will automatically be granted REFERENCES privileges on any columns that may be added to the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME in the future.
TRIGGER	The user has TRIGGER privilege on the table identified by TABLE_CATALOG, TABLE_SCHEMA, and TABLE_NAME.

5) The values of IS_GRANTABLE have the following meanings:

YES	The privilege being described was granted WITH GRANT OPTION and is thus grantable.
NO	The privilege being described was not granted WITH GRANT OPTION and is thus not grantable.

18.4.27 COLUMN_PRIVILEGES base table

Function

The COLUMN_PRIVILEGES table has one row for each column privilege descriptor. It effectively contains a representation of the column privilege descriptors.

Definition

```
CREATE TABLE COLUMN_PRIVILEGES
       (
       GRANTOR          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_CATALOG    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_NAME       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLUMN_NAME      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       PRIVILEGE_TYPE   INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT COLUMN_PRIVILEGE_TYPE_CHECK
           CHECK ( PRIVILEGE_TYPE IN ( 'SELECT', 'INSERT', 'UPDATE', 'REFERENCES' ) ),
       IS_GRANTABLE   INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT COLUMN_PRIVILEGE_IS_GRANTABLE_NOT_NULL NOT NULL
         CONSTRAINT COLUMN_PRIVILEGE_IS_GRANTABALE_CHECK
           CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT COLUMN_PRIVILEGE_PRIMARY_KEY
         PRIMARY KEY
           ( GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
             PRIVILEGE_TYPE, COLUMN_NAME ),

       CONSTRAINT COLUMN_PRIVILEGE_FOREIGN_KEY_COLUMNS
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME )
           REFERENCES COLUMNS,

       CONSTRAINT COLUMN_PRIVILEGE_GRANTOR_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,

       CONSTRAINT COLUMN_PRIVILEGE_GRANTEE_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTEE )
           REFERENCES USERS
       )
```

Description

1) The value of GRANTOR is the <authorization identifier> of the user who granted column privileges, on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME, to the user identified by the value of GRANTEE for the column privilege being described.

2) The value of GRANTEE is the <authorization identifier> of some user, or "PUBLIC" to indicate all users, to whom the column privilege being described is granted.

3) The values of TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the column to which the privilege being described was granted.

4) The values of PRIVILEGE_TYPE have the following meanings:

SELECT	The user has SELECT privilege on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
INSERT	The user has INSERT privilege on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
UPDATE	The user has UPDATE privilege on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.
REFERENCE	The user has REFERENCES privilege on the column identified by TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, and COLUMN_NAME.

5) The values of IS_GRANTABLE have the following meanings:

YES	The privilege being described was granted WITH GRANT OPTION and is thus grantable.
NO	The privilege being described was not granted WITH GRANT OPTION and is thus not grantable.

18.4.28 USAGE_PRIVILEGES base table

Function

The USAGE_PRIVILEGES table has one row for each usage privilege descriptor. It effectively contains a representation of the usage privilege descriptors.

Definition

```
CREATE TABLE USAGE_PRIVILEGES
       (
       GRANTOR        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       GRANTEE        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       OBJECT_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER,
       OBJECT_SCHEMA  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       OBJECT_NAME    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       OBJECT_TYPE    INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT USAGE_PRIVILEGES_OBJECT_TYPE_CHECK
           CHECK ( OBJECT_TYPE IN
                 ( 'DOMAIN', 'CHARACTER SET', 'COLLATION', 'TRANSLATION' ) ),
       IS_GRANTABLE   INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT USAGE_PRIVILEGES_IS_GRANTABLE_NOT_NULL NOT NULL
         CONSTRAINT USAGE_PRIVILEGES_IS_GRANTABLE_CHECK
           CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT USAGE_PRIVILEGES_PRIMARY_KEY
         PRIMARY KEY ( GRANTOR, GRANTEE, OBJECT_CATALOG, OBJECT_SCHEMA,
                       OBJECT_NAME, OBJECT_TYPE ),

       CONSTRAINT USAGE_PRIVILEGES_CHECK_REFERENCES_OBJECT
         CHECK ( ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME, OBJECT_TYPE ) IN
         ( SELECT DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,'DOMAIN'
             FROM DOMAINS
           UNION
           SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME,
                      'CHARACTER SET'
             FROM CHARACTER_SETS
           UNION
           SELECT COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, 'COLLATION'
             FROM COLLATIONS
           UNION
           SELECT TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME,
                      'TRANSLATION'
             FROM TRANSLATIONS
           UNION
           SELECT NULL_CLASS_CATALOG, NULL_CLASS_SCHEMA, NULL_CLASS_NAME, 'NULL CLASS'
             FROM NULL_CLASSES ) ),

       CONSTRAINT USAGE_PRIVILEGES_GRANTOR_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,

       CONSTRAINT USAGE_PRIVILEGES_GRANTEE_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTEE )
           REFERENCES USERS
       )
```

Description

1) The value of GRANTOR is the <authorization identifier> of the user who granted usage privileges, on the object of the type identified by OBJECT_TYPE that is identified by OBJECT_CATALOG, OBJECT_SCHEMA, and OBJECT_NAME, to the user identified by the value of GRANTEE for the usage privilege being described.

2) The value of GRANTEE is the <authorization identifier> of some user, or "PUBLIC" to indicate all users, to whom the usage privilege being described is granted.

3) The values of OBJECT_CATALOG, OBJECT_SCHEMA, and OBJECT_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the object to which the privilege applies.

4) The values of OBJECT_TYPE has the following meanings:

DOMAIN	The object to which the privilege applies is a domain.
CHARACTER SET	The object to which the privilege applies is a character set.
COLLATION	The object to which the privilege applies is a collation.
TRANSLATION	The object to which the privilege applies is a translation.
NULL CLASS	The object to which the privilege applies is a null class.

5) The values of IS_GRANTABLE have the following meanings:

YES	The privilege being described was granted WITH GRANT OPTION and is thus grantable.
NO	The privilege being described was not granted WITH GRANT OPTION and is thus not grantable.

18.4.29 CHARACTER_SETS base table

Function

The CHARACTER_SETS table has one row for each character set descriptor.

Definition

```
CREATE TABLE CHARACTER_SETS
       (
       CHARACTER_SET_CATALOG         INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CHARACTER_SET_SCHEMA          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CHARACTER_SET_NAME            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       FORM_OF_USE                   INFORMATION_SCHEMA.SQL_IDENTIFIER,
       NUMBER_OF_CHARACTERS          INFORMATION_SCHEMA.CARDINAL_NUMBER,
       DEFAULT_COLLATE_CATALOG       INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT CHARACTER_SETS_DEFAULT_COLLATE_CATALOG_NOT_NULL NOT NULL,
       DEFAULT_COLLATE_SCHEMA        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT CHARACTER_SETS_DEFAULT_COLLATE_SCHEMA_NOT_NULL NOT NULL,
       DEFAULT_COLLATE_NAME          INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT CHARACTER_SETS_DEFAULT_COLLATE_NAME_NOT_NULL NOT NULL,

       CONSTRAINT CHARACTER_SETS_PRIMARY_KEY
         PRIMARY KEY ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME ),


       CONSTRAINT CHARACTER_SETS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT CHARACTER_SETS_CHECK_REFERENCES_COLLATIONS
         CHECK ( DEFAULT_COLLATE_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                 ( DEFAULT_COLLATE_CATALOG, DEFAULT_COLLATE_SCHEMA,
                   DEFAULT_COLLATE_NAME ) IN
                     ( SELECT COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME
                       FROM COLLATIONS ) )
       )
```

Description

1) The values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the character set being described.

2) The value of FORM_OF_USE is the name of the form-of-use of the character set.

3) The value of NUMBER_OF_CHARACTERS is the number of characters in the character repertoire.

4) Case:

   a) If the default collation for the character repertoire is the order of characters in the repertoire, then the values of DEFAULT_COLLATE_CATALOG and DEFAULT_COLLATE_SCHEMA are the values of CHARACTER_SET_CATALOG and CHARACTER_SET_SCHEMA, respectively, and the value of DEFAULT_COLLATE_NAME is implementation-dependent.

b) Otherwise, the values of DEFAULT_COLLATE_CATALOG, DEFAULT_COLLATE_SCHEMA, and DEFAULT_COLLATE_NAME are catalog name, unqualified schema name, and qualified identifier, respectively, of the default collation.

5) There is a row in this table for the character set INFORMATION_SCHEMA.SQL_TEXT. In that row:

a) CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the name of the catalog, 'INFORMATION_SCHEMA', and 'SQL_TEXT', respectively.

b) FORM_OF_USE is implementation-defined.

c) NUMBER_OF_CHARACTERS is implementation-defined.

d) DEFAULT_COLLATE_CATALOG, DEFAULT_COLLATE_SCHEMA, and DEFAULT_COLLATE_NAME are the name of the catalog, 'INFORMATION_SCHEMA', and 'SQL_TEXT', respectively.

18.4.30 COLLATIONS base table

Function

The COLLATIONS table has one row for each character collation descriptor.

Definition

```
CREATE TABLE COLLATIONS
       (
       COLLATION_CATALOG INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_SCHEMA  INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_NAME    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CHARACTER_SET_CATALOG   INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT COLLATIONS_CHARACTER_SET_CATALOG_NOT_NULL NOT NULL,
       CHARACTER_SET_SCHEMA    INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT COLLATIONS_CHARACTER_SET_SCHEMA_NOT_NULL NOT NULL,
       CHARACTER_SET_NAME      INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT COLLATIONS_CHARACTER_SET_NAME_NOT_NULL NOT NULL,
       PAD_ATTRIBUTE           INFORMATION_SCHEMA.CHARACTER_DATA
           CONSTRAINT COLLATIONS_PAD_ATTRIBUTE_CHECK
             CHECK ( PAD_ATTRIBUTE IN ( 'NO PAD', 'PAD SPACE' ) ),
       COLLATION_TYPE    INFORMATION_SCHEMA.SQL_IDENTIFIER,
       COLLATION_DEFINITION    INFORMATION_SCHEMA.CHARACTER_DATA,
       COLLATION_DICTIONARY    INFORMATION_SCHEMA.CHARACTER_DATA,

       CONSTRAINT COLLATIONS_CHECK
         CHECK ( ( COLLATION_TYPE = 'DICTIONARY' AND
                   ( COLLATION_DEFINITION IS NULL AND COLLATION_DICTIONARY IS NOT NULL ) )

               OR
                   ( COLLATION_TYPE = 'SEQUENCE' AND
                   ( COLLATION_DICTIONARY IS NULL AND COLLATION_DEFINITION IS NOT NULL ) )

                 ),

       CONSTRAINT COLLATIONS_PAD_PRIMARY_KEY
         PRIMARY KEY ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ),

       CONSTRAINT COLLATIONS_PAD_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( COLLATION_CATALOG, COLLATION_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT COLLATIONS_CHECK_REFERENCES_CHARACTER_SETS
         CHECK ( CHARACTER_SET_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                 ( CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME )
                    IN
                      ( SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
                               CHARACTER_SET_NAME
                        FROM CHARACTER_SETS ) )
       )
```

Description

1) The values of COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the collation being described.

2) The values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the character set on which the collation is defined.

3) The values of PAD_ATTRIBUTE have the following meanings:

NO PAD	The collation being described has the NO PAD attribute.
PAD SPACE	The collation being described has the PAD SPACE attribute.

4) The values of COLLATION_TYPE have the following meanings:

DICTIONARY	The collation being described is a dictionary collation.
SEQUENCE	The collation being described is a collating sequence.

5) Case:

   a) If the value of COLLATION_TYPE is 'SEQUENCE', then the value of COLLATION_DEFINITION is a character representation of an <internal collation source> for the collation, as though the collation had been defined with an <internal collation source>.

   b) If the value of COLLATION_TYPE is 'DICTIONARY', then the value of COLLATION_DICTIONARY is an implementation-defined name of a collation dictionary.

6) There is a row in this table for the collation INFORMATION_SCHEMA.SQL_TEXT. That row contains the definition of the collation corresponding to the default collation for the characters in the character set SQL_TEXT. In that row:

   a) COLLATION_CATALOG, COLLATION_SCHEMA, and COLLATION_NAME are the name of the catalog, 'INFORMATION_SCHEMA', and 'SQL_TEXT', respectively.

   b) CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the name of the catalog, 'INFORMATION_SCHEMA', and 'SQL_TEXT', respectively.

   c) PAD_ATTRIBUTE is implementation-defined.

   d) COLLATION_TYPE is 'SEQUENCE'.

   e) COLLATION_DICTIONARY is null.

   f) COLLATION_DEFINITION is implementation-defined.

18.4.31 TRANSLATIONS base table

Function

The TRANSLATIONS table has one row for each character translation descriptor.

Definition

```
CREATE TABLE TRANSLATIONS
       (
       TRANSLATION_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TRANSLATION_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TRANSLATION_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SOURCE_CHARACTER_SET_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_SOURCE_CHARACTER_SET_CATALOG_NOT_NULL NOT NULL,
       SOURCE_CHARACTER_SET_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_SOURCE_CHARACTER_SET_SCHEMA_NOT_NULL NOT NULL,
       SOURCE_CHARACTER_SET_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_SOURCE_CHARACTER_SET_NAME_NOT_NULL NOT NULL,
       TARGET_CHARACTER_SET_CATALOG     INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_TARGET_CHARACTER_SET_CATALOG_NOT_NULL NOT NULL,
       TARGET_CHARACTER_SET_SCHEMA      INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_TARGET_CHARACTER_SET_SCHEMA_NOT_NULL NOT NULL,
       TARGET_CHARACTER_SET_NAME        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT TRANSLATIONS_TARGET_CHARACTER_SET_NAME_NOT_NULL NOT NULL,
       TRANSLATION_DEFINITION           INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TRANSLATION_DEFINITION_NOT_NULL NOT NULL,

       CONSTRAINT TRANSLATIONS_PRIMARY_KEY
         PRIMARY KEY ( TRANSLATION_CATALOG, TRANSLATION_SCHEMA, TRANSLATION_NAME ),

       CONSTRAINT TRANSLATIONS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( TRANSLATION_CATALOG, TRANSLATION_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT TRANSLATIONS_CHECK_REFERENCES_SOURCE
         CHECK ( SOURCE_CHARACTER_SET_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                 ( SOURCE_CHARACTER_SET_CATALOG, SOURCE_CHARACTER_SET_SCHEMA,
                   SOURCE_CHARACTER_SET_NAME ) IN
                     ( SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
                              CHARACTER_SET_NAME
                       FROM CHARACTER_SETS ) ),

       CONSTRAINT TRANSLATIONS_CHECK_REFERENCES_TARGET
         CHECK ( TARGET_CHARACTER_SET_CATALOG
                 <> ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
               OR
                 ( TARGET_CHARACTER_SET_CATALOG, TARGET_CHARACTER_SET_SCHEMA,
                     TARGET_CHARACTER_SET_NAME ) IN
                       ( SELECT CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA,
                                CHARACTER_SET_NAME
                         FROM CHARACTER_SETS ) )
       )
```

Description

1) The values of TRANSLATION_CATALOG, TRANSLATION_SCHEMA, and TRANSLATION_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the translation being described.

2) The values of SOURCE_CHARACTER_SET_CATALOG, SOURCE_CHARACTER_SET_SCHEMA, and SOURCE_CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the character set specified as the source for the translation.

3) The values of TARGET_CHARACTER_SET_CATALOG, TARGET_CHARACTER_SET_SCHEMA, and TARGET_CHARACTER_SET_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the character set specified as the target for the translation.

4) The value of TRANSLATION_DEFINITION is a character representation of an <internal translation source> for the translation, as though the translation had been defined with an <internal translation source>.

18.4.32 OPERATORS base table

Function

The OPERATORS table has one row for each operator.

Definition

```
CREATE TABLE OPERATORS
       (
         OPERATOR_CATALOG                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
         OPERATOR_SCHEMA                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         OPERATOR_NAME                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
         OPERATOR_LEVEL                      INFORMATION_SCHEMA.CARDINAL_NUMBER,
         OPERATOR_FORM                       INFORMATION_SCHEMA.CHARACTER_DATA,
```

- 2 columns deleted.

```
         CONSTRAINT OPERATORS_PRIMARY_KEY
           PRIMARY KEY ( OPERATOR_CATALOG, OPERATOR_SCHEMA, OPERATOR_NAME ),
```

- 1 constraint deleted.

```
         CONSTRAINT OPERATORS_FOREIGN_KEY_SCHEMATA
           FOREIGN KEY ( OPERATOR_CATALOG, OPERATOR_SCHEMA )
             REFERENCES SCHEMATA,

         CONSTRAINT VALID_FORMS
           CHECK ( OPERATOR_FORM  IN
                     ( 'INFIX ONLY',  'PREFIX ONLY',   'POSTFIX ONLY',
                       'PREFIX INFIX, 'PREFIX POSTFIX, 'INFIX POSTFIX ) )

         CONSTRAINT LEVEL_RANGE
           CHECK ( OPERATOR_LEVEL BETWEEN 10 AND 19 )
       )
```

Description

1) The value of TABLE_CATALOG is the name of the catalog in which the operator is defined.

2) The value of OPERATOR_NAME is the name of the operator.

3) The values of OPERATOR_FORM have the following meanings:

INFIX ONLY	The operator being described is infix only.
PREFIX ONLY	The operator being described is prefix only .
POSTFIX ONLY	The operator being described is postfix only.
PREFIX INFIX	The operator being described is prefix and infix.
PREFIX POSTFIX	The operator being described is prefix and postfix.

INFIX The operator being described is infix and postfix.
POSTFIX

4) The values of OPERATOR_LEVEL specify the level of the operator.

5) The value of ROUTINE_NAME is the name of the routine associated with the operator being described.

18.4.33 TRIGGERS base table

Function

The TRIGGERS base table has one row for each trigger. It effectively contains a representation of the trigger descriptors.

Definition

```
CREATE TABLE TRIGGERS
       (
         TRIGGER_CATALOG                    INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_SCHEMA                     INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_NAME                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
         EVENT_MANIPULATION                 INFORMATION_SCHEMA.CHARACTER_DATA,
           CONSTRAINT_TRIGGERS_EVENT_MANIPULATION_CHECK
             CHECK ( EVENT_MANIPULATION IN
                     ( 'INSERT', 'DELETE', 'UPDATE' ) ),
         EVENT_OBJECT_CATALOG               INFORMATION_SCHEMA.SQL_IDENTIFIER
           CONSTRAINT TRIGGERS_EVENT_OBJECT_CATALOG_NOT_NULL NOT
         EVENT_OBJECT_SCHEMA                INFORMATION_SCHEMA.SQL_IDENTIFIER
           CONSTRAINT TRIGGERS_EVENT_OBJECT_SCHEMA_NOT_NULL NOT N
         EVENT_OBJECT_TABLE                 INFORMATION_SCHEMA.SQL_IDENTIFIER
           CONSTRAINT TRIGGERS_EVENT_OBJECT_TABLE_NOT_NULL NOT NU
         EVENT_OBJECT_TABLE                 INFORMATION_SCHEMA.SQL_IDENTIFIER NOT NULL,

         ACTION_ORDER             INFORMATION_SCHEMA.CARDINAL_NUMBER NOT NULL,
         ACTION_CONDITION         INFORMATION_SCHEMA.CHARACTER_DATA,
         ACTION_STATEMENT_LIST    INFORMATION_SCHEMA.CHARACTER_DATA NOT NULL,
         ACTION_ORIENTATION       INFORMATION_SCHEMA.CHARACTER_DATA
             CHECK ( ACTION_ORIENTATION IN
                       ( 'ROW', 'STATEMENT' ) ),
         CONDITION_TIMING                   INFORMATION_SCHEMA.CHARACTER_DATA
             CHECK ( CONDITION_TIMING IN
                       ( 'BEFORE', 'AFTER' ) ),
         CONDITION_REFERENCE_OLD_TABLE      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         CONDITION_REFERENCE_NEW_TABLE      INFORMATION_SCHEMA.SQL_IDENTIFIER,
         COLUMN_LIST_IS_IMPLICIT            INFORMATION_SCHEMA.SQL_IDENTIFIER
           CONSTRAINT IS_IMPLICIT_CHECK
             CHECK ( COLUMN_LIST_IS_IMPLICIT IN ( 'YES', 'NO' )
                OR
                   COLUMN_LIST_IS_IMPLICIT IS NULL ),

         CONSTRAINT TRIGGERS_PRIMARY_KEY
           PRIMARY KEY ( TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME ),

         CONSTRAINT TRIGGERS_FOREIGN_KEY_SCHEMATA
           FOREIGN KEY ( TRIGGER_CATALOG, TRIGGER_SCHEMA )
             REFERENCES SCHEMATA,

         CONSTRAINT EVENT_MANIPULATION_UPDATE_CHECK
           CHECK ( ( EVENT_MANIPULATION <> 'UPDATE' AND
                     COLUMN_LIST_IS_IMPLICIT IS NULL )
                 OR
                   ( EVENT_MANIPULATION = 'UPDATE' AND
                     COLUMN_LIST_IS_IMPLICIT IS NOT NULL ) ),

         CONSTRAINT TRIGGERS_REFERENCES_TABLES
           CHECK ( EVENT_OBJECT_CATALOG  <>
                   ANY ( SELECT CATALOG_NAME FROM SCHEMATA )
                 OR
```

```
                    ( EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE )
                      IN
                        ( SELECT TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME
                            FROM TABLES ) )
    )
```

Description

1) The values of TRIGGER_CATALOG, TRIGGER_SCHEMA, and TRIGGER_NAME are the catalog name, schema name, and trigger name of the trigger being described.

2) The values of EVENT_MANIPULATION have the following meaning:

   INSERT The <trigger event> is INSERT.

   DELETE The <trigger event> is DELETE.

   UPDATE The <trigger event> is UPDATE.

3) The values of EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, and EVENT_OBJECT_TABLE are the qualified name of the <table name> of the trigger being described.

4) The values of CONDITION_TIMING have the following meaning:

   BEFORE The <trigger action time> is BEFORE.

   AFTER The <trigger action time> is AFTER.

5) The value of CONDITION_REFERENCE_OLD_TABLE is the <old value correlation name> of the trigger being described.

6) The value of CONDITION_REFERENCE_NEW_TABLE is the <new value correlation name> of the trigger being described.

7) The value of ACTION_ORDER is the ordinal position of the triggered in the list of triggers with the same EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, EVENT_OBJECT_TABLE, EVENT_MANIPULATION, CONDITION_TIMING, and ACTION_ORIENTATION.

8) The value of ACTION_CONDITION is a character representation of the <search condition> in the <triggered action> of the trigger being described.

9) ACTION_STATEMENT_LIST is a character representation of the <triggered SQL statement list> in the <triggered action> of the trigger being described.

10) The values of ACTION_ORIENTATION have the following meanings:

    ROW The <trigger action> specifies FOR EACH ROW.

    STATEMENT The <trigger action> specified FOR EACH STATEMENT.

11) The values of COLUMN_LIST_IS_IMPLICIT have the following meaning:

    YES The <trigger event> is UPDATE and the <trigger column list> is implicit.

    NO The <trigger event> is UPDATE and the <trigger column list> is explicit.

    null The <trigger event> is INSERT or DELETE.

18.4.34 TRIGGERED_COLUMNS base table

Function

The TRIGGERED_COLUMNS base table has one row for each column referenced by a trigger.

Definition

```
CREATE TABLE TRIGGERED_COLUMNS
       (
         TRIGGER_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_SCHEMA             INFORMATION_SCHEMA.SQL_IDENTIFIER,
         TRIGGER_NAME               INFORMATION_SCHEMA.SQL_IDENTIFIER,
         EVENT_OBJECT_CATALOG       INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT EVENT_OBJECT_CATALOG_NOT_NULL
           NOT NULL,
         EVENT_OBJECT_SCHEMA        INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT EVENT_OBJECT_SCHEMA_NOT_NULL
           NOT NULL,
         EVENT_OBJECT_TABLE         INFORMATION_SCHEMA.SQL_IDENTIFIER
         CONSTRAINT EVENT_OBJECT_TABLE_NOT_NULL
           NOT NULL,
         EVENT_OBJECT_COLUMN        INFORMATION_SCHEMA.SQL_IDENTIFIER,

         CONSTRAINT TRIGGERED_COLUMNS_PRIMARY_KEY     PRIMARY KEY
             ( TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME, EVENT_OBJECT_COLUMN ),

         CONSTRAINT TRIGGERED_COLUMNS_FOREIGN_KEY_TRIGGERS
           FOREIGN KEY ( TRIGGER_CATALOG, TRIGGER_SCHEMA, TRIGGER_NAME )
             REFERENCES TRIGGERS

         CONSTRAINT TRIGGERED_COLUMNS_FOREIGN_KEY_COLUMNS
           FOREIGN KEY
             ( EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA,
               EVENT_OBJECT_TABLE, EVENT_OBJECT_COLUMN )
             REFERENCES COLUMNS
       )
```

Description

1) The values of TRIGGER_CATALOG, TRIGGER_SCHEMA, and TRIGGER_NAME are the catalog name, schema name, and trigger name of the trigger being described.

2) The values of EVENT_OBJECT_CATALOG, EVENT_OBJECT_SCHEMA, and EVENT_OBJECT_TABLE are the catalog name, schema name, and table name of the table containing the column being described.

18.4.35 SQL_LANGUAGES base table

Function

The SQL_LANGUAGES table has one row for each ISO and implementation-defined SQL language binding and programming language for which conformance is claimed.

Definition

```
CREATE TABLE SQL_LANGUAGES
       (
         SQL_LANGUAGE_SOURCE                INFORMATION_SCHEMA.CHARACTER_DATA
           CONSTRAINT SQL_LANGUAGES_SOURCE_NOT_NULL NOT NULL,
         SQL_LANGUAGE_YEAR                  INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_CONFORMANCE           INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_INTEGRITY             INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_IMPLEMENTATION        INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_BINDING_STYLE         INFORMATION_SCHEMA.CHARACTER_DATA,
         SQL_LANGUAGE_PROGRAMMING_LANGUAGE INFORMATION_SCHEMA.CHARACTER_DATA,

       CONSTRAINT SQL_LANGUAGES_STANDARD_VALID_CHECK
         CHECK ( ( SQL_LANGUAGE_SOURCE = 'ISO 9075' AND
                   SQL_LANGUAGE_YEAR IS NOT NULL AND
                   SQL_LANGUAGE_CONFORMANCE IS NOT NULL AND
                   SQL_LANGUAGE_IMPLEMENTATION IS NULL AND
                   ( ( SQL_LANGUAGE_YEAR = '1987' AND
                       SQL_LANGUAGE_CONFORMANCE IN ( '1', '2' ) AND
                       SQL_LANGUAGE_INTEGRITY IS NULL AND
                       ( SQL_LANGUAGE_BINDING_STYLE = 'MODULE'
                         AND
                         SQL_LANGUAGE_PROGRAMMING_LANGUAGE IN
                           ( 'COBOL', 'FORTRAN', 'PASCAL', 'PLI' ) ) )
                     OR
                     ( SQL_LANGUAGE_YEAR = '1989' AND
                       SQL_LANGUAGE_CONFORMANCE IN ( '1', '2' ) AND
                       SQL_LANGUAGE_INTEGRITY IN ( 'NO', 'YES' ) AND
                       ( SQL_LANGUAGE_BINDING_STYLE = 'MODULE'
                         AND
                         SQL_LANGUAGE_PROGRAMMING_LANGUAGE IN
                             ( 'COBOL', 'FORTRAN', 'PASCAL', 'PLI' ) ) )
                     OR
                     ( SQL_LANGUAGE_YEAR = '1992' AND
                       SQL_LANGUAGE_CONFORMANCE IN
                           ( 'ENTRY', 'INTERMEDIATE', 'FULL' ) AND
                       SQL_LANGUAGE_INTEGRITY IS NULL AND
                       ( SQL_LANGUAGE_BINDING = 'MODULE'
                         AND
                         SQL_LANGUAGE_PROGRAMMING_LANGUAGE IN
                             ( 'ADA', 'C', 'COBOL',
                               'FORTRAN', 'MUMPS', 'PASCAL', 'PLI' ) ) ) ) )
                 OR
                 ( SQL_LANGUAGE_SOURCE <> 'ISO 9075' )
               )
       )
```

Description

1) Each row represents one binding of an ISO or implementation-defined SQL language to a standard module language.

2) The value of SQL_LANGUAGE_SOURCE is the name of the source of the language definition. The source of standard SQL language is the value 'ISO 9075', while the source of an implementation-defined version of SQL is implementation-defined.

3) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_YEAR is the year that the ISO standard was approved. Otherwise, the value of SQL_LANGUAGE_YEAR is implementation-defined.

   Note: As each new ISO SQL standard revision is approved, a new valid value of SQL_LANGUAGE_YEAR must be added to the CHECK constraint for this column.

4) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_CONFORMANCE is the conformance level to which conformance is claimed for the ISO standard. Otherwise, the value of SQL_LANGUAGE_CONFORMANCE is implementation-defined.

5) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075' and that language contains an optional integrity enhancement feature, then the value of SQL_LANGUAGE_INTEGRITY is 'YES' if conformance is claimed to the integrity enhancement feature, and 'NO' otherwise. Otherwise, the value of SQL_LANGUAGE_INTEGRITY is implementation-defined.

6) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_IMPLEMENTATION is null. Otherwise, the value of SQL_LANGUAGE_IMPLEMENTATION is an implementation-defined character string value.

7) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_BINDING_STYLE is the style of binding of the SQL language. If the value of SQL_LANGUAGE_BINDING_STYLE is 'MODULE', then the binding style of <module> is supported. Otherwise, the value of SQL_LANGUAGE_BINDING_STYLE is implementation-defined.

8) If the value of SQL_LANGUAGE_SOURCE is 'ISO 9075', then the value of SQL_LANGUAGE_PROGRAMMING_LANGUAGE is the programming language supported by the binding style indicated by the value of SQL_LANGUAGE_BINDING_STYLE. If the value of SQL_LANGUAGE_BINDING_STYLE is 'MODULE', then SQL_LANGUAGE_PROGRAMMING_LANGUAGE has the value 'ADA', 'C', 'COBOL', 'FORTRAN', 'MUMPS', 'PASCAL', or 'PLI'.

   Case:

   a) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'ADA', then Ada is supported with the given binding style.

   b) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'C', then C is supported with the given binding style.

   c) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'COBOL', then COBOL is supported with the given binding style.

   d) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'FORTRAN', then Fortran is supported with the given binding style.

   e) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'MUMPS', then MUMPS is supported with the given binding style.

f) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'PASCAL', then Pascal is supported with the given binding style.

g) If SQL_LANGUAGE_PROGRAMMING_LANGUAGE is 'PLI', then PL/I is supported with the given binding style.

Otherwise, the value of SQL_LANGUAGE_PROGRAMMING_LANGUAGE is implementation-defined.

18.4.36 USER_SESSION_ENABLED_ROLE base table

Function

Defines the role enabled by the <set role statement> for each SQL-session.

Definition

```
CREATE GLOBAL TEMPORARY TABLE USER_SESSION_ENABLED_ROLES
       (
       USER_NAME                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
       ENABLED_ROLE                    INFORMATION_SCHEMA.SQL_IDENTIFIER,

         PRIMARY KEY,

         FOREIGN KEY ( USER_NAME ) REFERENCES USERS,

         FOREIGN KEY ( ENABLED_ROLE ) REFERENCES ROLES
       )
```

Description

1) This table is a temporary table and is hence instantiated for each SQL-session.

2) When a <set role statement> is executed any existing rows are first deleted from the table and then a row is inserted.

3) USER_NAME identifies the <authorization identifier> of the current SQL-session.

4) ENABLED_ROLE identifies the <role name> enabled by the <set role statement>, if any.

ANSI Only–SQL3

18.4.37 DEFAULT_ROLE base table

Function

The DEFAULT_ROLE base table has one row for each default role descriptor. Each row effectively contains the default role name and <authorization identifier>.

Definition

```
CREATE TABLE DEFAULT_ROLE (
       DEFAULT_ROLE     INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT DEFAULT_ROLE_CHECK_DEFAULT_ROLE
         CHECK (DEFAULT_ROLE IN (SELECT ROLE_NAME FROM ROLES)),
       GRANTEE          INFORMATION_SCHEMA.SQL_IDENTIFIER,
       CONSTRAINT DEFAULT_ROLE_CHECK_GRANTEE
         CHECK (GRANTEE IN (SELECT USER_NAME FROM USERS) ),

       CONSTRAINT DEFAULT_ROLE_DESCRIPTORS_PRIMARY_KEY
         PRIMARY KEY (DEFAULT_ROLE, GRANTEE),
       CONSTRAINT DEFAULT_ROLE_DESCRIPTORS_FOREIGN_KEY_ROLES
         FOREIGN KEY ( ROLE_NAME)
         REFERENCES ROLES,
       CONSTRAINT DEFAULT_ROLE_DESCRIPTORS_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTEE )
         REFERENCES USERS
       )
```

Description

1) A row is inserted into this table whenever a <create default role> statement or <default role definition> is successfully executed. A row is deleted from this table whenever a <drop default role statement> is executed.

2) The value of the DEFAULT_ROLE is the <role name> of a <default role definition>.

3) The value of the USER_NAME is the current SQL-session <authorization identifier>, in which the <default role definition> statement is executed.

18.4.38 SUB_TABLES base table

Function

The table has one row for each supertable used to define a subtable.

Definition

```
CREATE TABLE SUB_TABLES (
       TABLE_CATALOG            INFORMATION_SCHEMA.SQL_IDENTIFIER,
       TABLE_SCHEMA             INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SUB_TABLE_NAME           INFORMATION_SCHEMA.SQL_IDENTIFIER,
       SUPER_TABLE_NAME         INFORMATION_SCHEMA.SQL_IDENTIFIER,

       CONSTRAINT SUB_TABLES_PRIMARY_KEY
         PRIMARY KEY
           ( TABLE_CATALOG, TABLE_SCHEMA, SUB_TABLE_NAME, SUPER_TABLE_NAME ),

       CONSTRAINT SUB_TABLES_FOREIGN_KEY_SUBTABLES
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, SUB_TABLE_NAME )
           REFERENCES TABLES,

       CONSTRAINT SUB_TABLES_FOREIGN_KEY_SUPERTABLES
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, SUPER_TABLE_NAME )
           REFERENCES TABLES )
```

Description

1) The means by which rows are inserted and deleted from this table is implementation-defined.

   Note: There is only one TABLE_CATALOG and TABLE_SCHEMA because of the Syntax Rules in Subclause 11.5, "<table definition>".

18.4.39 Assertions on the base tables

The following clauses specify assertions that apply to the base tables specified in Subclause 18.4, "Definition Schema".

The paramount criterion in formulating these assertions (after correctness) is ease of understanding for the human reader. There may well be formulations of the same assertions that are more efficient for some SQL-implementation, and quite possibly for all such implementations.

18.4.39.1 UNIQUE_CONSTRAINT_NAME assertion

Function

The UNIQUE_CONSTRAINT_NAME assertion ensures that the same combination of <schema name> and <constraint name> is not used by more than one constraint.

Note: The UNIQUE_CONSTRAINT_NAME assertion avoids the need for separate checks on DOMAINS, TABLE_CONSTRAINTS, and ASSERTIONS.

Definition

```
CREATE ASSERTION UNIQUE_CONSTRAINT_NAME
       CHECK ( 1 =
         ( SELECT MAX ( OCCURRENCES ) FROM
           ( SELECT COUNT (*) AS OCCURRENCES FROM
               ( SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                 FROM DOMAIN_CONSTRAINTS
               UNION ALL
               SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                 FROM TABLE_CONSTRAINTS
               UNION ALL
               SELECT CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
                 FROM ASSERTIONS )
             GROUP BY CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME
           )
         ) )
```

Description

1) The UNIQUE_CONSTRAINT_NAME assertion checks that no combination of (CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME) appears more than once in the tables DOMAINS, TABLE_CONSTRAINTS and ASSERTIONS.

18.4.39.2 EQUAL_KEY_DEGREES assertion

Function

The assertion EQUAL_KEY_DEGREES ensures that every foreign key is of the same degree as the corresponding unique constraint.

Definition

```
CREATE ASSERTION EQUAL_KEY_DEGREES
       CHECK
         ( NOT EXISTS
           ( SELECT * FROM       ( SELECT
                   COUNT ( DISTINCT FK.COLUMN_NAME ),
                   COUNT ( DISTINCT PK.COLUMN_NAME )
                 FROM KEY_COLUMN_USAGE AS FK,
                      REFERENTIAL_CONSTRAINTS AS RF,
                      KEY_COLUMN_USAGE AS PK
                 WHERE ( FK.CONSTRAINT_CATALOG, FK.CONSTRAINT_SCHEMA,
                           FK.CONSTRAINT_NAME )
                     = ( RF.CONSTRAINT_CATALOG, RF.CONSTRAINT_SCHEMA,
                           RF.CONSTRAINT_NAME )
                   AND
                       ( PK.CONSTRAINT_CATALOG, PK.CONSTRAINT_SCHEMA,
                           PK.CONSTRAINT_NAME )
                     = ( RF.UNIQUE_CONSTRAINT_CATALOG, RF.UNIQUE_CONSTRAINT_SCHEMA,
                           RF.UNIQUE_CONSTRAINT_NAME )
                 GROUP BY
                   RF.CONSTRAINT_CATALOG, RF.CONSTRAINT_SCHEMA, RF.CONSTRAINT_NAME
             ) AS REF ( FK_DEGREE, PK_DEGREE )
               WHERE FK_DEGREE <> PK_DEGREE ) )
```

18.4.39.3 KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 assertion

Function

The assertion KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 ensures that every unique or primary key constraint has at least one unique column and that every referential constraint has at least one referencing column.

Definition

```
CREATE ASSERTION KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1
       CHECK
         ( NOT EXISTS
           ( SELECT * FROM
               TABLE_CONSTRAINTS
               FULL OUTER JOIN
                 KEY_COLUMN_USAGE
                   USING ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME )
               WHERE COLUMN_NAME IS NULL
                   AND CONSTRAINT_TYPE IN
                           ( 'UNIQUE', 'PRIMARY KEY', 'FOREIGN KEY' ) ) )
```

19 Status codes

19.1 SQLSTATE

The character string value returned in an SQLSTATE parameter comprises a 2-character class value followed by a 3-character subclass value, each with an implementation-defined character set that has a one-octet form-of-use and is restricted to <digit>s and <simple Latin upper case letter>s. Table 20, "SQLSTATE class and subclass values", specifies the class value for each condition and the subclass value or values for each class value.

Class values that begin with one of the <digit>s '0', '1', '2', '3', or '4' or one of the <simple Latin upper case letter>s 'A', 'B', 'C', 'D', 'E', 'F', 'G', or 'H' are returned only for conditions defined in this
ANSI American
ISO International
Standard or in any other
ANSI American or
International Standard. The class value 'HZ' is reserved for conditions defined in ISO/IEC DIS 9579-2. Subclass values associated with such classes that also begin with one of those 13 characters are returned only for conditions defined in this
ANSI American
ISO International
Standard; subclass values associated with such classes that begin with one of the <digit>s '5', '6', '7', '8', or '9' or one of the <simple Latin upper case letter>s 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', or 'Z' are reserved for implementation-specified conditions and are called *implementation-defined subclasses*.

Class values that begin with one of the <digit>s '5', '6', '7', '8', or '9' or one of the <simple Latin upper case letter>s 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', or 'Z' are reserved for implementation-specified exception conditions and are called *implementation-defined classes*. All subclass values except '000', which means *no subclass*, associated with such classes are reserved for implementation-specified conditions and are called *implementation-defined subclasses*. An implementation-defined completion condition shall be indicated by returning an implementation-defined subclass in conjunction with one of the classes *successful completion*, *warning*, or *no data*.

If a subclass value is not specified for a condition, then either subclass '000' or an implementation-defined subclass is returned.

If multiple completion conditions: *warning* or multiple exception conditions, including implementation-defined exception conditions, are raised, then it is implementation-dependent which of the corresponding SQLSTATE values is returned in the SQLSTATE status parameter, provided that the precedence rules in Subclause 4.32.1, "Status parameters", are obeyed. Any number of applicable conditions values in addition to the one returned in the SQLSTATE status parameter, may be returned in the diagnostics area.

An implementation-specified condition may duplicate, in whole or in part, a condition defined in this
ANSI American
ISO International
Standard; however, if such a condition occurs as a result of executing a statement, then the corresponding implementation-defined SQLSTATE value must not be returned in the SQLSTATE parameter but may be returned in the diagnostics area.

The "Category" column has the following meanings: "S" means that the class value given corresponds to successful completion and is a completion condition; "W" means that the class value given corresponds to a successful completion but with a warning and is a completion condition; "N" means that the class value given corresponds to a no-data situation and is a completion condition; "X" means that the class value given corresponds to an exception condition.

Table 20—SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
X	ambiguous cursor name	3C	(*no subclass*)	000
X	asynchronous SQL statement not accepted	05	(*no subclass*)	000
X	asynchronous SQL statement returned an SQLSTATE value	06	(*no subclass*)	000
ANSI Only–SQL3				
X	attempt to select from an empty multiset	A0	(*no subclass*)	000
X	attempt to select from an empty set	A1	(*no subclass*)	000
X	cardinality violation	21	(*no subclass*)	000
X	connection exception	08	(*no subclass*)	000
			connection does not exist	003
			connection failure	006
			connection name in use	002
			SQL-client unable to establish SQL-connection	001
			SQL-server rejected establishment of SQL-connection	004
			transaction resolution unknown	007
X	cursor sensitivity exception	36	(*no subclass*)	000
			request failed	002
			request rejected	001
X	data exception	22	(*no subclass*)	000
			character not in repertoire	021
			datetime field overflow	008
			division by zero	012
			error in assignment	005
			indicator overflow	022
ANSI Only–SQL3				

Table 20—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
			insertion point not unique	028
			interval field overflow	015
			invalid character value for cast	018
			invalid datetime format	007
			invalid enumeration name	017
			invalid enumeration value	016
			invalid escape character	019
			invalid escape sequence	025
			invalid indicator parameter value	010
			invalid limit value	020
			invalid parameter value	023
			invalid regular expression	01B
			invalid time zone displacement value	009
			invalid update value	014
			null instance in attribute reference	028
			null row not permitted in table	01C
			null value, no indicator parameter	002
			numeric value out of range	003
			row already exists	028
			string data, length mismatch	026
			string data, right truncation	001
			substring error	011
			trim error	027
			unterminated C string	024
X	dependent privilege descriptors still exist	2B	(*no subclass*)	000
X	duplicate asynchronous SQL statement identifier	41	(*no subclass*)	000
ANSI Only–SQL3				
X	empty list passed to HEAD	A2	(*no subclass*)	000

Table 20—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
X	external function call exception	39	(*no subclass*)	000
			invalid SQLSTATE returned	001
			null value not allowed	002
X	external function exception	38	(*no subclass*)	000
			invalid return value for predicate	002
			not available	001
X	feature not supported	0A	(*no subclass*)	000
			multiple server transactions	001
X	integrity constraint violation	23	(*no subclass*)	000
			restrict violation	001
X	invalid ADT instance	3G	(*no subclass*)	000
X	invalid authorization specification	28	(*no subclass*)	000
X	invalid catalog name	3D	(*no subclass*)	000
X	invalid character set name	2C	(*no subclass*)	000
X	invalid condition number	35	(*no subclass*)	000
X	invalid connection name	2E	(*no subclass*)	000
X	invalid cursor mode state	29	(*no subclass*)	000
X	invalid cursor name	34	(*no subclass*)	000
X	invalid cursor state	24	(*no subclass*)	000
			cascade off incompatible with holdable-cursor	001
ISO Only–SQL3				
X	invalid role name	0E	(*no subclass*)	000
X	invalid schema name	3F	(*no subclass*)	000
X	invalid SQL descriptor name	33	(*no subclass*)	000
X	invalid SQL statement name	26	(*no subclass*)	000
X	invalid SQL statement	30	(*no subclass*)	000
X	invalid target specification value	31	(*no subclass*)	000
X	invalid target type specification	0D	(*no subclass*)	000
X	invalid transaction initiation	0B	(*no subclass*)	000
X	invalid transaction state	25	(*no subclass*)	000
			active SQL-transaction	001

Table 20—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
			branch transaction already active	002
			held cursor requires same isolation level	008
			inappropriate access mode for branch transaction	003
			inappropriate isolation level for branch transaction	004
			no active SQL-transaction for branch transaction	005
			read-only SQL-transaction	006
			schema and data statement mixing not supported	007
X	invalid transaction termination	2D	(*no subclass*)	000
X	locator exception	0F	(*no subclass*)	000
			invalid specification	001
N	no data	02	(*no subclass*)	000
X	no outstanding asynchronous SQL statement	04	(*no subclass*)	000

ANSI Only–SQL3

X	non-single element list CAST to element type	A3	(*no subclass*)	000
X	non-single element multiset CAST to element type	A4	(*no subclass*)	000
X	non-single element set CAST to element type	A5	(*no subclass*)	000

X	Remote Database Access	HZ	(*See Table 22, "SQLSTATE Subclasses for Class 'HZ'", for the definition of protocol subconditions and subclass code values*)	
X	savepoint exception	3B	(*no subclass*)	000
			invalid specification	001
			too many	002
X	SQL statement not yet complete	03	(*no subclass*)	000
S	successful completion	00	(*no subclass*)	000
X	syntax error or access rule violation	42	(*no subclass*)	000

ANSI Only–SQL3

Table 20—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
X	syntax error or access rule violation in SQL_Table function	45	(*no subclass*)	000
X	transaction rollback	40	(*no subclass*)	000
			integrity constraint violation	002
			serialization failure	001
			statement completion unknown	003
			triggered action exception	004
X	triggered action exception	09	(*no subclass*)	000
X	triggered data change violation	27	(*no subclass*)	000
ANSI Only–SQL3				
X	unable to determine element type for LIST	A6	(*no subclass*)	000
X	unable to determine element type for MULTISET	A7	(*no subclass*)	000
X	unable to determine element type for SET	A8	(*no subclass*)	000
W	warning	01	(*no subclass*)	000
			cursor operation conflict	001
			default value too long for information schema	00B
			disconnect error	002
			external routine warning (the value of *xx* to be chosen by the author of the external routine)	H*xx*
			implicit zero-bit padding	008
			null value eliminated in set function	003
			privilege not granted	007
			privilege not revoked	006
			query expression too long for information schema	00A
			search condition too long for information schema	009
			string data, right truncation	004

Table 20—SQLSTATE class and subclass values (Cont.)

Category	Condition	Class	Subcondition	Subclass
X	with check option violation	44	(*no subclass*)	000

19.2 SQLCODE

Table 21, "SQLCODE values", specifies the integer value returned in an SQLCODE parameter for each condition. The negative values that indicate exception conditions are implementation-defined.

Table 21—SQLCODE values

Value	Condition
0	successful completion or warning
+100	no data
$+m$[1]	warning
$-n$	exception
[1] m greater than 0 and not equal to 100	

Note: SQLSTATE is the preferred status parameter. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this
ANSI American
ISO International
Standard. See Annex D, "Deprecated features".

19.3 Remote Database Access SQLSTATE Subclasses

This International Standard reserves SQLSTATE class 'HZ' for Remote Database Access errors, which may occur when an SQL-client interacts with an SQL-server across a communications network using an RDA Application Context. ISO/IEC 9579-1, ISO/IEC 9579-2, ISO 8649, and ISO/IEC 10026-2 define a number of exception conditions that must be detected in a conforming ISO RDA implementation. This Subclause defines SQLSTATE subclass codes for each such condition out of the set of codes reserved for International Standards.

If an implementation using RDA reports a condition shown in Table 22, "SQLSTATE Subclasses for Class 'HZ'", for a given exception condition, then it shall use the SQLSTATE class code 'HZ' and the subclass codes shown, and shall set the values of CLASS_ORIGIN to 'ISO 9075' and SUBCLASS_ ORIGIN as indicated in Table 22, "SQLSTATE Subclasses for Class 'HZ'", when those exceptions are retrieved by a <get diagnostics statement>.

An implementation using client-server communications other than RDA may report conditions corresponding to the conditions shown in Table 22, "SQLSTATE Subclasses for Class 'HZ'", using the SQLSTATE class code 'HZ' and the corresponding subclass codes shown. It may set the values of CLASS_ORIGIN to 'ISO 9075' and SUBCLASS_ORIGIN as indicated in Table 22, "SQLSTATE Subclasses for Class 'HZ'". Any other communications error shall be returned with a subclass code from the implementation-defined range, with CLASS_ORIGIN set to 'ISO 9075' and SUBCLASS_ ORIGIN set to an implementation-defined character string.

A Remote Database Access exception may also result in an SQL completion condition defined in Table 20, "SQLSTATE class and subclass values" (such as '40000', *transaction rollback*); if such a condition occurs, then the 'HZ' class SQLSTATE shall not be returned in the SQLSTATE parameter, but may be returned in the Diagnostics Area.

Table 22—SQLSTATE Subclasses for Class 'HZ'

RDA Generic Condition	SQLSTATE Subclass	Subclass Origin
(*no subclass*)	000	ISO/IEC 9579-1
Access Control Violation	010	ISO/IEC 9579-1
Bad Repetition Count	020	ISO/IEC 9579-1
Command Handle Unknown	030	ISO/IEC 9579-1
Control Authentication Failure	040	ISO/IEC 9579-1
Control Services Not Allowed	230	ISO/IEC 9579-1
Data Resource Handle Not Specified	050	ISO/IEC 9579-1
Data Resource Handle Unknown	060	ISO/IEC 9579-1
Data Resource Name Not Specified	070	ISO/IEC 9579-1
Data Resource Not Available (Permanent)	080	ISO/IEC 9579-1
Data Resource Not Available (Transient)	081	ISO/IEC 9579-1
Data Resource Already Open	090	ISO/IEC 9579-1
Data Resource Unknown	100	ISO/IEC 9579-1
Dialogue ID Unknown	110	ISO/IEC 9579-1
Duplicate Command Handle	120	ISO/IEC 9579-1
Duplicate Data Resource Handle	130	ISO/IEC 9579-1
Duplicate Dialogue ID	140	ISO/IEC 9579-1
Duplicate Operation ID	150	ISO/IEC 9579-1
Invalid Sequence	160	ISO/IEC 9579-1
Dialogue Already Active[1]	161	ISO/IEC 9579-1
Dialogue Initializing[1]	162	ISO/IEC 9579-1
Dialogue Not Active[1]	163	ISO/IEC 9579-1
Dialogue Terminating[1]	164	ISO/IEC 9579-1
Transaction Not Open[1]	165	ISO/IEC 9579-1
Transaction Open[1]	166	ISO/IEC 9579-1
Transaction Terminating[1]	167	ISO/IEC 9579-1
No Data Resource Available	170	ISO/IEC 9579-1
Operation Aborted (Permanent)	180	ISO/IEC 9579-1
Operation Aborted (Transient)	181	ISO/IEC 9579-1
Operation Cancelled	190	ISO/IEC 9579-1
Service Not Negotiated	200	ISO/IEC 9579-1
Transaction Rolled Back	210	ISO/IEC 9579-1

[1]Subcomponent of Invalid Sequence

(Continued on next page)

Table 22—SQLSTATE Subclasses for Class 'HZ' (Cont.)

RDA Generic Condition	SQLSTATE Subclass	Subclass Origin
User Authentication Failure	220	ISO/IEC 9579-1
Host Identifier Error	300	ISO/IEC 9579-2
Invalid SQL Conformance Level	310	ISO/IEC 9579-2
RDA Transaction Not Open	320	ISO/IEC 9579-2
RDA Transaction Open	325	ISO/IEC 9579-2
SQL Access Control Violation	330	ISO/IEC 9579-2
SQL Database Resource Already Open	340	ISO/IEC 9579-2
SQL DBL Argument Count Mismatch	350	ISO/IEC 9579-2
SQL DBL Argument Type Mismatch	360	ISO/IEC 9579-2
SQL DBL No Character Set	365	ISO/IEC 9579-2
SQL DBL Transaction Statement Not Allowed	370	ISO/IEC 9579-2
SQL Usage Mode Violation	380	ISO/IEC 9579-2
Abort Failure Service Provider	410	ISO 8649
Abort Failure Service User	411	ISO 8649
Associate Failure (Permanent)	420	ISO 8649
Associate Failure (Transient)	421	ISO 8649
Release Failure	430	ISO 8649
Begin Dialogue Rejected Provider	450	ISO/IEC 10026-2
Begin Dialogue Rejected User	451	ISO/IEC 10026-2
Heuristic Hazard	460	ISO/IEC 10026-2
Heuristic Mix	461	ISO/IEC 10026-2
PAbort Rollback False	470	ISO/IEC 10026-2
PAbort Rollback True	471	ISO/IEC 10026-2
Rollback	480	ISO/IEC 10026-2
UAbort Rollback False	490	ISO/IEC 10026-2
UAbort Rollback True	491	ISO/IEC 10026-2
UError	4A0	ISO/IEC 10026-2

Note: The subclass conditions ("RDA Generic Conditions") shown in the first column of this table correspond to ASN.1 error types; the condition names were formed by eliminating blanks between individual words of ther ASN.1 error types and, in some cases, selecting one of a set of enumerated subparameters.

20 Conformance

20.1 Introduction

This
ANSI American
ISO International
Standard specifies conforming SQL language and conforming SQL-implementations.

Conforming SQL language shall abide by the BNF Format, associated Syntax Rules and Access Rules, definitions, and descriptions.

A conforming SQL-implementation shall process conforming SQL language according to the associated General Rules, definitions, and descriptions.

The object identifier for Database Language SQL is specified in Subclause 3.4, "Object identifier for Database Language SQL".

20.2 Claims of conformance

Claims of conformance to this
ANSI American
ISO International
Standard shall state:

1) Which *level of conformance* is claimed:

   a) Full SQL (The complete database language specified in this
      ANSI American
      ISO International
      Standard.)

   b) Intermediate SQL (Intermediate SQL is a subset of Full SQL as specified in the Leveling Rules.)

   c) Entry SQL (Entry SQL is a subset of Intermediate SQL as specified in the Leveling Rules.)

2) Whether or not the Module (<module>) *binding style* is supported.

3) For the module binding style, which of the following programming languages are supported:

   a) Ada

   b) C

   c) COBOL

   d) Fortran

   e) MUMPS

f) Pascal

g) PL/I

4) The definitions for all elements and actions that this Standard specifies as implementation-defined.

20.3 Extensions and options

A conforming implementation may provide additional facilities or options not specified by the level of this
ANSI American
ISO International
Standard to which conformance is claimed. This may imply an implementation-defined extension of the list of reserved words (<reserved word>) and thereby may prevent proper processing of some programs that otherwise meet the requirements of this
ANSI American
ISO International
Standard.

An implementation remains conforming even if it provides user options to process nonconforming SQL language or to process conforming SQL language in a nonconforming manner.

20.4 Flagger requirements

Implementations that claim conformance only to Entry SQL may, but are not required to, provide an SQL Flagger (see Subclause 4.45, "SQL Flagger").

Implementations that claim conformance to Intermediate SQL shall provide an SQL Flagger (see Subclause 4.45, "SQL Flagger") that supports the following “level of flagging” options:

— Entry SQL Flagging

— Intermediate SQL Flagging

and the following “extent of checking” option:

— Syntax Only

Implementations that claim conformance to Full SQL shall provide an SQL Flagger (see Subclause 4.45, "SQL Flagger") that supports the following “level of flagging” options:

— Entry SQL Flagging

— Intermediate SQL Flagging

— Full SQL Flagging

and the following “extent of checking” options:

— Syntax Only

— Catalog Lookup

Annex A
(Informative)

Leveling the SQL Language

This Annex describes the restrictions placed on conforming Full SQL, Intermediate SQL and Entry SQL language.

A.1 Full SQL Specifications

1) Subclause 5.3, "<literal>":

   a) A <general literal> shall not be an <enumeration literal>.

   b) A <general literal> shall not be a <boolean literal>.

   c) A <general literal> shall not be an <oid literal>.

2) Subclause 5.4, "Names and identifiers":

   a) Conforming Full SQL language shall not contain any <component name>.

   b) Conforming Full SQL language shall not contain any <abstract data type name>.

   c) Conforming Full SQL language shall not contain any <savepoint name>.

   d) Conforming Full SQL language shall not contain any <role name>.

   e) Conforming Full SQL language shall not contain any <null class name>.

   f) Conforming Full SQL language shall not contain any <external routine name>.

   ISO Only–SQL3

   g) Conforming Full SQL language shall not contain any <query name>.

   h) Conforming Full SQL language shall not contain any <operator name>.

   i) Conforming Full SQL language shall not contain any <type template name>.

j) Conforming Full SQL language shall not contain any <trigger name>.

k) Conforming Full SQL language shall not contain any <enumeration name>.

3) Subclause 6.1, "<data type>":

a) A <data type> shall not be a <user-defined type>,
ANSI <template parameter name>, or <row identifier type>.
ISO or <template parameter name>.

b) A <predefined type> shall not be an <enumerated type> or <boolean type>.

c) A <data type> shall not be a <collection type>.

d) A <data type> shall not be a <binary large object string type>.

e) A <character string type> shall not specify CHARACTER LARGE OBJECT, CHAR LARGE OBJECT, or CLOB.

f) A <national character string type> shall not specify NATIONAL CHARACTER LARGE OBJECT, NCHAR LARGE OBJECT, or NCLOB.

4) Subclause 6.2, "<value specification> and <target specification>":

a) A <general value specification> shall not be a <template parameter name>, <function name>, <item reference>, <component reference>, or CURRENT_PATH.

b) A <simple value specification> shall not be an an <item reference>.

c) A <target specification> shall not be a <template parameter name> or <component reference>.

5) Subclause 6.16, "<subtype treatment>":

a) Conforming Full SQL Language shall contain no <subtype treatment>.

6) Subclause 6.3, "<item reference>":

a) An <item reference> that contains an <item qualifier> shall be a column reference.

ANSI Only–SQL3

b) The <item reference> shall not identify an implicit row identifier column.

7) Subclause 6.5, "<component reference>":

a) Conforming Full SQL language shall contain no <component reference>.

8) Subclause 6.7, "<table reference>":

ANSI Only–SQL3

a) A <table specification> shall be a <table name>.

9) Subclause 6.9, "<numeric value function>":

a) A <position expression> shall not be a <blob position expression>.

10) Subclause 6.10, "<string value function>":

a) A <string value function> shall not be a <regular expression substring function>, a <blob value function>, or a <character overlay function>.

11) Subclause 6.15, "<value expression>":

a) A <value expression> shall not be an <enumerated value expression>.

b) A <value expression> shall not be a <boolean value expression>.

c) A <value expression> shall not specify a <collection value expression>.

d) A <value expression primary> that is a <table subquery> shall satisfy the Syntax Rules and General Rules for a <scalar subquery>.

e) A <value expression> shall not be a <distinct type value expression>.

f) A <value expression primary> shall not be a <subtype treatment>.

12) Subclause 6.19, "<enumerated value expression>":

a) Conforming Full SQL language shall contain no <enumerated value expression>.

13) Subclause 6.18, "<string value expression>":

a) A <string value expression> shall not be a <blob value expression>.

14) Subclause 6.22, "<operator expression>":

a) Conforming Full SQL language shall not contain an <operator expression>.

15) Subclause 6.23, "<boolean value expression>":

a) Conforming Full SQL language shall contain no <boolean value expression>.

16) Subclause 7.1, "<row value constructor>":

a) A <null specification> shall contain no <null state>.

17) Subclause 7.4, "<set value constructor>":

a) Conforming Full SQL language shall contain no <set value constructor>.

18) Subclause 7.5, "<multiset value constructor>":

a) Conforming Full SQL language shall contain no <multiset value constructor>.

19) Subclause 7.6, "<list value constructor>":

a) Conforming Full SQL language shall contain no <list value constructor>.

20) Subclause 7.9, "<joined table>":

a) A <qualified join> shall contain no <constraint join>.

21) Subclause 7.13, "<query specification>":

ISO Only—caused by ANSI changes not yet considered by ISO

a) A <query specification> shall contain no <user-defined updatability clause> or <referencing new values clause>.

ANSI Only—caused by ISO changes not yet considered by ANSI

b) A <derived column> shall contain no <user-defined column update clause>.

c) A query specification is not inherently updatable if other than a column reference or if a <column reference> appears more than once.

22) Subclause 7.14, "<query expression>":

a) A <non-join query term> shall contain no <recursive union>.

ISO Only—caused by ANSI changes not yet considered by ISO

b) A <query expression> shall contain no <user-defined updatability clause> or <referencing values clause>.

23) Subclause 7.15, "<recursive union>":

a) Conforming Full SQL language shall contain no <recursive union>.

24) Subclause 8.1, "<predicate>":

a) Conforming Full SQL language shall contain no <similar predicate>.

b) Conforming Full SQL language shall contain no <quantified predicate>.

ANSI Only—caused by ISO changes not yet considered by ANSI

c) Conforming Full SQL language shall contain no <there is predicate>.

d) Conforming Full SQL language shall contain no <function invocation predicate>.

25) Subclause 8.6, "<similar predicate>":

a) A <like predicate> shall not be an <octet like predicate>.

26) Subclause 8.6, "<similar predicate>":

a) Conforming Full SQL language shall contain no <similar predicate>.

27) Subclause 8.7, "<null predicate>":

a) A <null predicate> shall contain no <null values specification>.

28) Subclause 8.13, "<quantified predicate>":

a) Conforming Full SQL language shall contain no <quantified predicate>.

ANSI Only—caused by ISO changes not yet considered by ANSI

29) Subclause 8.14, "<there is predicate>":

a) Conforming Full SQL language shall contain no <there is predicate>.

30) Subclause 8.16, "<boolean predicate>":

a) Conforming Intermediate SQL language shall not specify a <boolean predicate>.

31) Subclause 8.18, "<search condition>":

a) A <boolean primary> shall not specify a <boolean value expression>.

32) Subclause 10.3, "<generated type reference>":

a) Conforming Full SQL language shall contain no <generated type reference>.

33) Subclause 10.4, "<privileges>":

a) An <action> shall not specify EXECUTE, TRIGGER, or UNDER.

b) An <action> that specifies SELECT shall not contain a <privilege column list>.

c) An <object name> shall not specify DATA TYPE or TYPE TEMPLATE.

34) Subclause 11.1, "<schema definition>":

a) Conforming Full SQL language shall not contain any <null class definition>.

b) Conforming Full SQL language shall not contain any <trigger definition>.

c) Conforming Full SQL language shall not contain any <external function declaration>.

- 1 list element moved to Part 4

d) Conforming Full SQL language shall not contain any <abstract data type definition>.

e) Conforming Full SQL language shall not contain any <distinct type definition>.

f) Conforming Full SQL language shall not contain any <type template definition>.

g) Conforming Full SQL language shall not contain any <role definition>.

h) Conforming Full SQL language shall not contain any <grant role statement>.

i) Conforming Full SQL language shall not contain any <schema path specification>.

35) Subclause 11.2, "<alter schema statement>":

a) Conforming Full SQL language shall not contain an <alter schema statement>.

36) Subclause 11.3, "<add operators definition>":

a) Conforming Full SQL language shall not contain an <add operators definition>.

37) Subclause 11.5, "<table definition>":

ANSI Only—caused by ISO changes not yet considered by ANSI

a) A <table definition> shall not specify a <table type>.

ANSI Only–SQL3

b) A <table definition> shall not specify WITH IDENTITY.

c) A <table definition> shall not specify a <constant or updatable>.

d) A <table definition> shall not specify a <subtable clause>.

e) A <table element> shall not be a <like clause>.

38) Subclause 11.6, "<column definition>":

a) A <column definition> shall not contain a <null clause>.

39) Subclause 11.7, "<attribute definition>":

a) An <attribute definition> shall not be specified.

40) Subclause 11.9, "<default clause>":

a) A <default option> that specifies NULL shall not contain a <null state>.

b) A <default option> shall not specify CURRENT_PATH.

41) Subclause 11.11, "<unique constraint definition>":

a) A <unique constraint definition> shall specify a <unique column list>.

42) Subclause 11.12, "<referential constraint definition>":

a) A <references specification> shall not contain PENDANT.

b) If a <referenced table and columns> specifies a <left paren>, then it shall also specify a <reference column list>.

c) A <referential action> shall not be RESTRICT.

ISO Only—caused by ANSI changes not yet considered by ISO

d) A <referential action> shall not be NO ACTION.

e) The data type of each referencing column shall be the same as the data type of the corresponding referenced column.

43) Subclause 11.14, "<alter table statement>":

a) An <alter table action> shall not be an <add supertable clause>.

b) An <alter table action> shall not be a <drop supertable clause>.

44) Subclause 11.16, "<alter column definition>":

a) Conforming Full SQL language shall not contain any <drop column domain clause>.

45) Subclause 11.19, "<drop column domain clause>":

a) Conforming Full SQL language shall not contain any <drop column domain clause>.

46) Subclause 11.21, "<add supertable clause>":

a) Conforming Full SQL language shall contain no <add supertable clause>.

47) Subclause 11.22, "<drop supertable clause>":

a) Conforming Full SQL language shall contain no <drop supertable clause>.

48) Subclause 11.28, "<domain definition>":

a) A <domain definition> shall not contain a <null clause>.

49) Subclause 11.33, "<drop domain constraint definition>":

a) Conforming Full SQL language shall contain no <constraint disposition>.

50) Subclause 11.34, "<drop domain statement>":

a) Conforming Full SQL language shall not contain any <constraint disposition>.

51) Subclause 11.35, "<null class definition>":

a) Conforming Full SQL language shall contain no <null class definition>.

52) Subclause 11.36, "<drop null class statement>":

a) Conforming Full SQL language shall contain no <drop null class statement>.

53) Subclause 11.37, "<character set definition>":

a) A <character set source> shall not contain a <plus sign>.

b) A <character set source> shall not contain a <character list>.

c) A <character set definition> shall not contain a <form-of-use specification>.

54) Subclause 11.39, "<collation definition>":

a) A <collation source> shall not be a <collation dictionary specification>.

b) A <collation source> shall not be a <collation routine specification>.

c) A <collation sequence definition> shall not be an <internal collation source>.

55) Subclause 11.41, "<translation definition>":

a) A <translation source> shall not be a <translation routine>.

b) A <translation specification> shall not be an <internal translation source>.

56) Subclause 11.43, "<assertion definition>":

a) An <assertion definition> shall contain no <assertion trigger>.

b) A <triggered assertion> shall not specify FOR.

57) Subclause 11.45, "<trigger definition>":

a) Conforming Full SQL language shall contain no <trigger definition>.

58) Subclause 11.46, "<drop trigger statement>":

a) Conforming Full SQL language shall contain no <drop trigger statement>.

- 1 list element moved to Part 4

59) Subclause 11.47, "<abstract data type definition>":

a) Conforming Full SQL language shall contain no <abstract data type definition>.

60) Subclause 11.48, "<abstract data type body>":

a) Conforming Full SQL language shall contain no <abstract data type body>.

61) Subclause 11.49, "<distinct type definition>":

a) Conforming Full SQL language shall contain no <distinct type definition>.

62) Subclause 11.50, "<type template definition>":

a) Conforming Full SQL language shall contain no <type template definition>.

63) Subclause 11.51, "<drop type template statement>":

a) Conforming Full SQL language shall contain no <drop type template statement>.

64) Subclause 11.52, "<drop data type statement>":

a) Conforming Full SQL language shall contain no <drop data type statement>.

65) Subclause 11.53, "<operators definition>":

a) Conforming Full SQL language shall not contain an <operators definition>.

66) Subclause 11.54, "<grant statement>":

ANSI Only–SQL3

a) In Conforming Full SQL language, an <object name> shall not specify SET, or LIST.

ANSI Only–SQL3

67) Subclause 11.60, "<default role definition>":

a) Conforming Full SQL language shall contain no <default role definition>.

68) Subclause 11.61, "<drop default role statement>":

a) Conforming Full SQL language shall contain no <drop default role statement>.

69) Subclause 11.55, "<role definition>":

a) Conforming Full SQL language shall contain no <role definition>.

70) Subclause 11.56, "<grant role statement>":

a) Conforming Full SQL language shall contain no <grant role statement>.

71) Subclause 11.57, "<revoke role statement>":

a) Conforming Full SQL language shall contain no <revoke role statement>.

72) Subclause 11.58, "<drop role statement>":

a) Conforming Full SQL language shall contain no <drop role statement>.

73) Subclause 12.2, "<module>":

a) A <module contents> shall not specify a <temporary view declaration>, or a <routine> that specifies an external routine.

b) Conforming Full SQL language shall not contain any <module path specification>.

- 1 list element moved to Part 4

74) Subclause 12.3, "<SQL procedure statement>":

a) An <SQL procedure statement> shall not specify ASYNC.

b) An <SQL schema definition statement> shall not be a <role definition>, <grant role statement>, <null class definition>, <trigger definition>, <abstract data type definition>, <distinct type definition>, or <type template definition>.

c) An <SQL shema manipulation statement> shall not be a <revoke role statement>, <drop role statement>, <drop null class statement>, <drop trigger statement>, <drop data type statement>, or <drop type template statement>.

d) An <SQL transaction statement> shall not be a <test completion statement>, <savepoint statement>, or <release savepoint statement>.

e) An <SQL session statement> shall not be a <set role statement>.

f) An <SQL transaction statement> shall not be a <start transaction statement>.

75) Subclause 13.1, "<declare cursor>":

a) A <declare cursor> shall not specify SENSITIVE.

b) A <declare cursor> shall not specify WITH HOLD.

76) Subclause 13.2, "<open statement>":

a) An <open statement> shall not contain an <open cascade option>.

77) Subclause 13.8, "<insert statement>":

ANSI Only—caused by ISO changes not yet considered by ANSI

a) An <insert statement> shall not specify an <insert point> or a <cursor name>.

78) Subclause 13.9, "<update statement: positioned>":

a) An <update statement: positioned> shall contain a <table name>.

b) An <update statement: positioned> shall not contain an <update type>.

c) An <update target> shall not contain an <object column list>.

79) Subclause 13.10, "<update statement: searched>":

a) An <update statement: searched> shall not contain an <update type>.

b) An <update target> shall not contain an <object column list>.

ANSI Only—caused by ISO changes not yet considered by ANSI

c) An <update statement: searched> shall not contain an <update by moving>.

- 1 list entry deleted

80) Subclause 13.12, "<temporary abstract data type declaration>":

a) Conforming Full SQL language shall not contain any <temporary abstract data type declaration>.

81) Subclause 13.13, "<temporary view declaration>":

a) Conforming Full SQL language shall contain no <temporary view declaration>.

82) Subclause 14.1, "<start transaction statement>":

a) Conforming Full SQL language shall not contain any <start transaction statement>.

83) Subclause 14.2, "<set transaction statement>":

a) Conforming Full SQL language shall not specify LOCAL.

84) Subclause 14.4, "<test completion statement>":

a) Conforming Full SQL language shall contain no <test completion statement>.

85) Subclause 14.5, "<savepoint statement>":

a) Conforming Full SQL language shall contain no <savepoint statement>.

86) Subclause 14.6, "<release savepoint statement>":

a) Conforming Full SQL language shall contain no <release savepoint statement>.

87) Subclause 14.7, "<commit statement>":

a) Conforming Full SQL language shall not specify CHAIN.

88) Subclause 14.8, "<rollback statement>":

a) A <rollback statement> shall contain no <savepoint clause>.

b) Conforming Full SQL language shall not specify CHAIN.

89) Subclause 16.3, "<set role statement>":

a) Conforming Full SQL language shall contain no <set role statement>.

90) Subclause 17.1, "<get diagnostics statement>":

a) Conforming Full SQL language shall not specify a <statement information item name> that is COMMAND_FUNCTION_CODE or DYNAMIC_FUNCTION_CODE.

b) Conforming Full SQL language shall not specify a <statement information item name> that is TRIGGER_CATALOG, TRIGGER_SCHEMA, or TRIGGER_NAME.

c) Conforming Full SQL language shall not specify a <statement information item name> that is ROUTINE_CATALOG, ROUTINE_SCHEMA, ROUTINE_NAME, or SPECIFIC_NAME.

91) Subclause 18.2.32, "OPERATORS view":

a) Conforming Full SQL language shall not reference the OPERATORS view.

92) Subclause 18.2.33, "TRIGGERS view":

a) Conforming Full SQL language shall not reference the TRIGGERS view.

93) Subclause 18.2.34, "TRIGGERED_COLUMNS view":

a) Conforming Full SQL language shall not reference the TRIGGERED_COLUMNS view.

94) Subclause 18.2.35, "ENABLED_APPLICABLE_ROLES view":

a) Conforming Full SQL language shall not reference the ENABLED_APPLICABLE_ROLES view.

95) Subclause 18.2.36, "ROLE_TABLE_GRANTS view":

a) Conforming Full SQL language shall not reference the ROLE_TABLE_GRANTS view.

96) Subclause 18.2.37, "ROLE_COLUMN_GRANTS view":

a) Conforming Full SQL language shall not reference the ROLE_COLUMN_GRANTS view.

97) Subclause 18.2.38, "APPLICABLE_ROLES view":

a) Conforming Full SQL language shall not reference the APPLICABLE_ROLES view.

98) Subclause 18.2.39, "ADMINISTRABLE_ROLE_AUTHORIZATIONS view":

a) Conforming Full SQL language shall not reference the ADMINISTRABLE_ROLE_AUTHORIZATIONS view.

99) Subclause 18.3, "USER_SESSION_ENABLED_ROLE view":

a) Conforming Full SQL language shall not reference the USER_SESSION_ENABLED_ROLE view.

A.2 Intermediate SQL Specifications

1) All Full SQL specifications are included as Intermediate SQL specifications.

2) Subclause 5.2, "<token> and <separator>":

   a) No <identifier body> shall end in an <underscore>.

3) Subclause 5.3, "<literal>":

   a) An <unsigned integer> that is a <seconds fraction> shall not contain more than 6 <digit>s.

   b) A <general literal> shall not be a <bit string literal> or a <hex string literal>.

4) Subclause 5.4, "Names and identifiers":

   a) Conforming Intermediate SQL language shall not contain any explicit <catalog name>, <connection name>, <collation name>, <translation name>, <form-of-use conversion name>, or <qualified local table name>.

5) Subclause 6.1, "<data type>":

   a) A <datetime type> shall not specify a <time precision> or <timestamp precision>.

   b) A <data type> shall not be a <bit string type>.

6) Subclause 6.7, "<table reference>":

   a) A <table reference> shall not be a <derived table>.

7) Subclause 6.8, "<set function specification>":

   a) If a <general set function> specifies DISTINCT, then the <value expression> shall be a column reference.

8) Subclause 6.9, "<numeric value function>":

   a) A <numeric value function> shall not be a <position expression>.

   b) A <numeric value function> shall not contain a <length expression> that is a <bit length expression>.

9) Subclause 6.10, "<string value function>":

   a) A <character value function> shall not be a <fold>.

   b) Conforming Intermediate SQL language shall contain no <character translation>.

   c) Conforming Intermediate SQL language shall contain no <form-of-use conversion>.

   d) Conforming Intermediate SQL language shall contain no <bit value function>.

10) Subclause 6.11, "<datetime value function>":

   a) Conforming Intermediate SQL language shall contain no <time precision> or <timestamp precision>.

11) Subclause 6.18, "<string value expression>":

    a) Conforming Intermediate SQL language shall not contain any <collate clause>.

    b) Conforming Intermediate SQL language shall contain no <bit value expression>.

12) Subclause 7.1, "<row value constructor>":

    a) A <row value constructor> that is not simply contained in a <table value constructor> or an <overlaps predicate> shall not contain more than one <row value constructor element>.

    b) A <row value constructor> shall not be a <row subquery>.

13) Subclause 7.3, "<table value constructor>":

    a) A <table value constructor> shall contain exactly one <row value constructor> that shall be of the form "(<row value constructor list>)".

    b) A <table value constructor> shall be the <query expression> of an <insert statement>.

14) Subclause 7.9, "<joined table>":

    a) Conforming Intermediate SQL language shall contain no <cross join>.

    b) Conforming Intermediate SQL language shall not specify UNION JOIN.

15) Subclause 7.11, "<group by clause>":

    a) Conforming Intermediate SQL language shall not contain any <collate clause>.

16) Subclause 7.13, "<query specification>":

    a) The <set quantifier> DISTINCT shall not be specified more than once in a <query specification>, excluding any <subquery> of that <query specification>.

17) Subclause 7.14, "<query expression>":

    a) A <simple table> shall not be a <table value constructor> except in an <insert statement>.

    b) Conforming Intermediate SQL shall contain no <explicit table>.

18) Subclause 8.1, "<predicate>":

    a) A <predicate> shall not be a <match predicate>.

19) Subclause 8.4, "<in predicate>":

    a) Conforming Intermediate SQL language shall not contain a <value expression> in an <in value list> that is not a <value specification>.

20) Subclause 8.11, "<match predicate>":

    a) Conforming Intermediate SQL language shall not contain any <match predicate>.

21) Subclause 8.18, "<search condition>":

    a) A <boolean test> shall not specify a <truth value>.

22) Subclause 10.4, "<privileges>":

a) An <action> that specifies INSERT shall not contain a <privilege column list>.

23) Subclause 10.6, "<collate clause>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

24) Subclause 10.7, "<constraint name definition> and <constraint attributes>":

a) Conforming Intermediate SQL language shall contain no explicit <constraint attributes>.
Note: This means that INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

25) Subclause 11.1, "<schema definition>":

a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

b) Conforming Intermediate SQL language shall not contain any <collation definition>.

c) Conforming Intermediate SQL language shall not contain any <translation definition>.

26) Subclause 11.5, "<table definition>":

a) Conforming Intermediate SQL language shall not specify TEMPORARY and shall not reference any global or local temporary table.

27) Subclause 11.6, "<column definition>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

28) Subclause 11.7, "<attribute definition>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

29) Subclause 11.12, "<referential constraint definition>":

a) A <references specification> shall not specify MATCH.

b) A <referential triggered action> shall not contain an <update rule>.

c) The order of the column names in a <reference column list> shall be the same as the order of column names of the corresponding unique constraint of the referenced table.

30) Subclause 11.13, "<check constraint definition>":

a) The <search condition> contained in a <check constraint definition> shall not contain a <subquery>.

b) The REFERENCES privilege is not required for <check constraint definition> access.

31) Subclause 11.26, "<view definition>":

a) Conforming Intermediate SQL language shall not contain any <levels clause>, but the effect shall be that defined for a <levels clause> of CASCADED.

32) Subclause 11.28, "<domain definition>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

33) Subclause 11.29, "<alter domain statement>":

a) Conforming Intermediate SQL language shall contain no <alter domain statement>.

34) Subclause 11.30, "<set domain default clause>":

a) Conforming Intermediate SQL language shall contain no <set domain default clause>.

35) Subclause 11.31, "<drop domain default clause>":

a) Conforming Intermediate SQL language shall contain no <drop domain default clause>.

36) Subclause 11.32, "<add domain constraint definition>":

a) Conforming Intermediate SQL language shall contain no <add domain constraint definition>.

37) Subclause 11.33, "<drop domain constraint definition>":

a) Conforming Intermediate SQL language shall contain no <drop domain constraint definition>.

38) Subclause 11.37, "<character set definition>":

a) In conforming Intermediate SQL language, <collation source> shall specify DEFAULT.

39) Subclause 11.39, "<collation definition>":

a) Conforming Intermediate SQL language shall not contain any <collation definition>.

40) Subclause 11.40, "<drop collation statement>":

a) Conforming Intermediate SQL language shall not contain any <drop collation statement>.

41) Subclause 11.41, "<translation definition>":

a) Conforming Intermediate SQL language shall contain no <translation definition>.

42) Subclause 11.42, "<drop translation statement>":

a) Conforming Intermediate SQL language shall contain no <drop translation statement>.

43) Subclause 11.43, "<assertion definition>":

a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

44) Subclause 11.44, "<drop assertion statement>":

a) Conforming Intermediate SQL language shall not contain any <drop assertion statement>.

45) Subclause 11.54, "<grant statement>":

a) In Conforming Intermediate SQL language, an <object name> shall not specify COLLATION or TRANSLATION.

46) Subclause 12.2, "<module>":

a) A <module> shall not contain a <temporary table declaration>.

47) Subclause 13.1, "<declare cursor>":

a) A <declare cursor> shall not specify INSENSITIVE.

b) If an <updatability clause> of FOR UPDATE with or without a <column name list> is specified, then neither SCROLL nor ORDER BY shall be specified.

48) Subclause 13.7, "<delete statement: searched>":

a) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in the <search condition>.

49) Subclause 13.8, "<insert statement>":

a) The leaf generally underlying table of *T* shall not be generally contained in the <query expression> immediately contained in the <insert columns and source> except as the <item qualifier> of a column reference.

50) Subclause 13.9, "<update statement: positioned>":

a) *CR* shall not be an ordered cursor.

51) Subclause 13.10, "<update statement: searched>":

a) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in the <search condition> or in any <value expression> immediately contained in any <update source> contained in the <set clause list>.

52) Subclause 13.11, "<temporary table declaration>":

a) Conforming Intermediate SQL language shall not contain any <temporary table declaration>.

53) Subclause 14.3, "<set constraints mode statement>":

a) Conforming Intermediate SQL language shall not contain any <set constraints mode statement>.

54) Subclause 15.1, "<connect statement>":

a) Conforming Intermediate SQL language shall not contain any <connect statement>.

55) Subclause 15.2, "<set connection statement>":

a) Conforming Intermediate SQL language shall not contain any <set connection statement>.

56) Subclause 15.3, "<disconnect statement>":

a) Conforming Intermediate SQL language shall not contain any <disconnect statement>.

57) Subclause 18.2.25, "COLLATIONS view":

a) Conforming Intermediate SQL language shall not reference the COLLATIONS view.

58) Subclause 18.2.26, "TRANSLATIONS view":

a) Conforming Intermediate SQL language shall not reference the TRANSLATIONS view.

A.3 Entry SQL Specifications

1) All Intermediate SQL specifications are included as Entry SQL specifications.

2) Subclause 5.2, "<token> and <separator>":

   a) No <regular identifier> or <delimited identifier body> shall contain more than 18 <character representation>s.

   b) An <identifier body> shall contain no <simple Latin lower case letter>.

3) Subclause 5.3, "<literal>":

   a) A <general literal> shall not be a <national character string literal>.

   b) A <general literal> shall not be a <datetime literal> or <interval literal>.

   c) A <character string literal> shall contain at least one <character representation>.

   d) Conforming Entry SQL language shall contain exactly one repetition of <character representation> (that is, it shall contain exactly one sequence of “<quote> <character representation>... <quote>”).

   e) A <character string literal> shall not specify a <character set specification>.

4) Subclause 5.4, "Names and identifiers":

   a) Conforming Entry SQL language shall not contain any <domain name>, <constraint name>, or <character set name>.

   b) An <identifier> shall not specify a <character set specification>.

   c) A <parameter name> shall immediate contain a <colon>.

5) Subclause 6.1, "<data type>":

   a) A <character string type> shall not specify VARYING or VARCHAR.

   b) A <data type> shall not be a <datetime type> or an <interval type>.

   c) A <data type> shall not be a <national character string type> nor specify CHARACTER SET.

6) Subclause 6.2, "<value specification> and <target specification>":

   a) A <general value specification> shall not specify VALUE.

   b) A <general value specification> shall not specify CURRENT_USER, SYSTEM_USER, or SESSION_USER.

      Note: Although CURRENT_USER and USER are semantically the same, in Entry SQL, CURRENT_USER must be specified as USER.

7) Subclause 6.7, "<table reference>":

   a) A <table reference> shall not be a <joined table>.

b) The optional <key word> AS shall not be specified.

c) <derived column list> shall not be specified.

8) Subclause 6.8, "<set function specification>":

a) If a <general set function> specifies or implies ALL, then COUNT shall not be specified.

b) If a <general set function> specifies or implies ALL, then the <value expression> shall include a column reference that references a column of *T*.

c) If the <value expression> contains a column reference that is an outer reference, then the <value expression> shall be a <column reference>.

d) No column reference contained in a <set function specification> shall reference a column derived from a <value expression> that generally contains a <set function specification>.

9) Subclause 6.9, "<numeric value function>":

a) A <numeric value function> shall not be a <length expression>.

b) A <numeric value function> shall not be an <extract expression>.

10) Subclause 6.10, "<string value function>":

a) A <character value function> shall not be a <character substring function>.

b) A <character value function> shall not be a <trim function>.

11) Subclause 6.11, "<datetime value function>":

a) Conforming Entry SQL language shall not contain any <datetime value function>.

12) Subclause 6.13, "<case expression>":

a) Conforming Entry SQL language shall not contain any <case expression>.

13) Subclause 6.14, "<cast specification>":

a) Conforming Entry SQL language shall not contain any <cast specification>.

14) Subclause 6.15, "<value expression>":

a) A <value expression> shall not be a <datetime value expression>.

b) A <value expression> shall not be an <interval value expression>.

c) A <value expression primary> shall not be a <case expression>.

d) A <value expression primary> shall not be a <cast specification>.

e) A <value expression primary> shall not be a <scalar subquery> except when the <value expression primary> is simply contained in a <value expression> that is simply contained in the second <row value constructor> of a <comparison predicate>.

15) Subclause 6.18, "<string value expression>":

a) A <character value expression> shall not be a <concatenation>.

16) Subclause 6.20, "<datetime value expression>":

a) Conforming Entry SQL language shall not contain any <datetime value expression>.

17) Subclause 6.21, "<interval value expression>":

a) Conforming Entry SQL language shall not contain any <interval value expression>.

18) Subclause 7.1, "<row value constructor>":

a) A <row value constructor element> shall not specify DEFAULT.

19) Subclause 7.7, "<table expression>":

a) If the table identified in the <from clause> is a grouped view, then the <table expression> shall not contain a <where clause>, <group by clause>, or <having clause>.

20) Subclause 7.8, "<from clause>":

a) If the table identified by <table reference> is a grouped view, then the <from clause> shall contain exactly one <table reference>.

21) Subclause 7.9, "<joined table>":

a) Conforming Entry SQL language shall not contain any <joined table>.

22) Subclause 7.10, "<where clause>":

a) A <value expression> directly contained in the <search condition> shall not include a reference to a column that generally contains a <set function specification>.

23) Subclause 7.13, "<query specification>":

a) A <query specification> is not inherently updatable if the <where clause> of the <table expression> contains a <subquery>.

b) A <select sublist> shall be a <derived column>.

c) If the <table expression> of the <query specification> is a grouped view, then the <select list> shall not contain a <set function specification>.

24) Subclause 7.14, "<query expression>":

a) A <query expression> shall not specify EXCEPT.

b) A <query term> shall not specify INTERSECT.

c) A <query expression> shall not contain a <joined table>.

d) A <query expression> shall not specify CORRESPONDING.

e) If UNION is specified, then except for column names, the descriptors of the first and second operands shall be identical and the descriptor of the result is identical to the descriptor of the operands.

25) Subclause 7.16, "<scalar subquery>, <row subquery>, and <table subquery>":

a) If a <subquery> is contained in a <comparison predicate>, then the <table expression> in the <query specification> shall not contain a <group by clause> or a <having clause> and shall not identify a grouped view.

b) The <query expression> contained in a <subquery> shall be a <query specification>.

c) If a <table subquery> is simply contained in an <exists predicate>, then the <select list> of the <query specification> directly contained in the <table subquery> shall comprise either an <asterisk> or a single <derived column>.

26) Subclause 8.1, "<predicate>":

a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

b) Conforming Entry SQL language shall not contain any <unique predicate>.

c) Conforming Entry SQL language shall not contain any <distinct predicate>.

27) Subclause 8.5, "<like predicate>":

a) The <character match value> shall be a column reference.

b) A <pattern> shall be a <value specification>.

c) An <escape character> shall be a <value specification>.

28) Subclause 8.7, "<null predicate>":

a) A <row value constructor> shall be a column reference.

29) Subclause 8.10, "<unique predicate>":

a) Conforming Entry SQL language shall not contain any <unique predicate>.

30) Subclause 8.12, "<overlaps predicate>":

a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

31) Subclause 8.15, "<distinct predicate>":

a) Conforming Entry SQL language shall not contain any <distinct predicate>.

32) Subclause 10.1, "<interval qualifier>":

a) Conforming Entry SQL language shall not contain any <interval qualifier>.

33) Subclause 10.2, "<language clause>":

a) A <language clause> shall not specify MUMPS.

34) Subclause 10.5, "<character set specification>":

a) Conforming Entry SQL language shall not contain a <character set specification>.

35) Subclause 10.7, "<constraint name definition> and <constraint attributes>":

a) Conforming Entry SQL language shall contain no <constraint name definition>.

36) Subclause 11.1, "<schema definition>":

a) Conforming Entry SQL language shall not contain any <domain definition>.

b) A <schema name clause> shall specify AUTHORIZATION and shall not specify a <schema name>.

c) A <schema character set specification> shall not be specified.

d) Conforming Entry SQL language shall not contain any <character set definition>.

37) Subclause 11.4, "<drop schema statement>":

a) Conforming Entry SQL language shall not contain a <drop schema statement>.

38) Subclause 11.6, "<column definition>":

a) A <column definition> shall not contain a <domain name>.

b) A <column constraint> shall not contain a <referential triggered action>.

c) Conforming Entry SQL language shall not contain any <constraint name definition>.

39) Subclause 11.9, "<default clause>":

a) A <default option> shall not specify a <datetime value function>, SYSTEM_USER, SESSION_USER, or CURRENT_USER.

40) Subclause 11.10, "<table constraint definition>":

a) Conforming Entry SQL language shall contain no <constraint name definition>.

41) Subclause 11.11, "<unique constraint definition>":

a) If PRIMARY KEY or UNIQUE is specified, then the <column definition> for each column whose <column name> is in the <unique column list> shall specify NOT NULL.

42) Subclause 11.12, "<referential constraint definition>":

a) A <referential constraint definition> shall not contain a <referential triggered action>.

43) Subclause 11.14, "<alter table statement>":

a) Conforming Entry SQL language shall not contain an <alter table statement>.

44) Subclause 11.15, "<add column definition>":

a) Conforming Entry SQL language shall not contain an <add column definition>.

45) Subclause 11.16, "<alter column definition>":

a) Conforming Entry SQL language shall not contain an <alter column definition>.

46) Subclause 11.17, "<set column default clause>":

a) Conforming Entry SQL language shall not contain a <set column default clause>.

47) Subclause 11.18, "<drop column default clause>":

a) Conforming Entry SQL language shall not contain a <drop column default clause>.

48) Subclause 11.20, "<drop column definition>":

a) Conforming Entry SQL language shall not contain a <drop column definition>.

49) Subclause 11.21, "<add supertable clause>":

a) Conforming Entry SQL language shall not contain an <add table constraint definition>.

50) Subclause 11.22, "<drop supertable clause>":

a) Conforming Entry SQL language shall not contain a <drop table constraint definition>.

51) Subclause 11.23, "<add table constraint definition>":

a) Conforming Entry SQL language shall not contain an <add table constraint definition>.

52) Subclause 11.24, "<drop table constraint definition>":

a) Conforming Entry SQL language shall not contain a <drop table constraint definition>.

53) Subclause 11.25, "<drop table statement>":

a) Conforming Entry SQL language shall not contain any <drop table statement>.

54) Subclause 11.26, "<view definition>":

a) The <query expression> in a <view definition> shall be a <query specification>.

55) Subclause 11.27, "<drop view statement>":

a) Conforming Entry SQL language shall not contain a <drop view statement>.

56) Subclause 11.28, "<domain definition>":

a) Conforming Entry SQL language shall not contain any <domain definition>.

57) Subclause 11.35, "<null class definition>":

a) Conforming Entry SQL language shall not contain a <drop domain statement>.

58) Subclause 11.37, "<character set definition>":

a) Conforming Entry SQL language shall not specify any <character set definition>.

59) Subclause 11.38, "<drop character set statement>":

a) Conforming Entry SQL language shall contain no <drop character set statement>.

60) Subclause 11.54, "<grant statement>":

a) In Conforming Entry SQL language, an <object name> shall not specify TABLE.

b) In Conforming Entry SQL language, an <object name> shall not specify CHARACTER SET or DOMAIN.

61) Subclause 11.59, "<revoke statement>":

a) Conforming Entry SQL language shall not contain a <revoke statement>.

62) Subclause 12.1, "<SQL-client module definition>":

a) An SQL-client <module> shall be associated with an SQL-agent during its execution. An SQL-agent shall be associated with at most one SQL-client module.

63) Subclause 12.2, "<module>":

a) A <module character set specification> shall not be specified.

b) A <module authorization clause> shall specify AUTHORIZATION and shall not specify SCHEMA.

64) Subclause 12.3, "<SQL procedure statement>":

a) An <SQL procedure statement> shall not be an <SQL schema definition statement>.

65) Subclause 13.1, "<declare cursor>":

a) A <declare cursor> shall not specify SCROLL.

b) A <cursor specification> shall not contain an <updatability clause>.

66) Subclause 13.3, "<fetch statement>":

a) A <fetch statement> shall not contain a <fetch orientation>.

b) A <fetch statement> shall not specify FROM.

c) If the data type of the target identified by the *i*-th <target specification> in the <fetch target list> is an exact numeric type, then the data type of the *i*-th column of the table *T* shall be an exact numeric type.

67) Subclause 13.5, "<select statement: single row>":

a) If the data type of the target identified by the *i*-th <target specification> in the <select target list> is an exact numeric type, then the data type of the *i*-th column of the table *T* shall be an exact numeric type.

b) The <table expression> shall not include a <group by clause> or a <having clause> and shall not identify a grouped view.

68) Subclause 13.8, "<insert statement>":

a) The <query expression> that is contained in an <insert columns and source> shall be a <query specification> or it shall be a <table value constructor> that contains exactly one <row value constructor> of the form “<left paren> <row value constructor list> <right paren>”, and each <row value constructor element> of that <row value constructor list> shall be a <value specification> or a <null specification>.

b) If the data type of the target identified by the *i*-th <column name> is an exact numeric type, then the data type of the *i*-th item of the <insert statement> shall be an exact numeric type.

c) If the data type of the target *C* identified by the *i*-th <column name> is character string, then the length in characters of the *i*-th item of the <insert statement> shall be less than or equal to the length of *C*.

d) The <insert columns and source> shall immediately contain a <query expression>.

69) Subclause 13.9, "<update statement: positioned>":

a) If the data type of the column identified by the *i*-th <object column> is an exact numeric type, then the data type of the *i*-th <value expression> in the <update statement: positioned> shall be an exact numeric type.

b) If the data type of the column identified by the *i*-th <object column> *C* is character string, then the length in characters of the *i*-th <value expression> in the <update statement: positioned> shall be less than or equal to the length of *C*.

c) An <update source> shall not specify DEFAULT.

70) Subclause 13.10, "<update statement: searched>":

a) If the data type of the column identified by the *i*-th <object column> is an exact numeric type, then the data type of the *i*-th <value expression> in the <update statement: searched> shall be an exact numeric type.

b) If the data type of the column identified by the *i*-th <object column> *C* is character string, then the length in characters of the *i*-th <value expression> in the <update statement: searched> shall be less than or equal to the length of *C*.

71) Subclause 14.2, "<set transaction statement>":

a) Conforming Entry SQL language shall not contain any <set transaction statement>.

72) Subclause 14.7, "<commit statement>":

a) In conforming Entry SQL language, WORK shall be specified.

73) Subclause 14.8, "<rollback statement>":

a) In conforming Entry SQL language, WORK shall be specified.

74) Subclause 16.2, "<set session authorization identifier statement>":

a) Conforming Entry SQL language shall not contain any <set session authorization identifier statement>.

75) Subclause 16.4, "<set local time zone statement>":

a) Conforming Entry SQL language shall not contain any <set local time zone statement>.

76) Subclause 17.1, "<get diagnostics statement>":

a) Conforming Entry SQL language shall not contain any <get diagnostics statement>.

77) Subclause 18.2.1, "INFORMATION_SCHEMA Schema":

a) Conforming Entry SQL language shall not reference the Information Schema.

78) Subclause 18.2.2, "INFORMATION_SCHEMA_CATALOG_NAME base table":

a) Conforming Entry SQL language shall not reference the Information Schema.

79) Subclause 18.2.3, "INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY assertion":

a) Conforming Entry SQL language shall not reference the Information Schema.

80) Subclause 18.2.4, "SCHEMATA view":

a) Conforming Entry SQL language shall not reference the Information Schema.

81) Subclause 18.2.5, "DOMAINS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

82) Subclause 18.2.6, "DOMAIN_CONSTRAINTS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

83) Subclause 18.2.7, "TABLES view":

a) Conforming Entry SQL language shall not reference the Information Schema.

84) Subclause 18.2.8, "VIEWS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

85) Subclause 18.2.9, "COLUMNS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

86) Subclause 18.2.11, "NULL_CLASSES view":

a) Conforming Entry SQL language shall not reference the Information Schema.

87) Subclause 18.2.12, "NULL_STATES view":

a) Conforming Entry SQL language shall not reference the Information Schema.

88) Subclause 18.2.13, "TABLE_PRIVILEGES view":

a) Conforming Entry SQL language shall not reference the Information Schema.

89) Subclause 18.2.14, "COLUMN_PRIVILEGES view":

a) Conforming Entry SQL language shall not reference the Information Schema.

90) Subclause 18.2.15, "USAGE_PRIVILEGES view":

a) Conforming Entry SQL language shall not reference the Information Schema.

91) Subclause 18.2.19, "TABLE_CONSTRAINTS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

92) Subclause 18.2.20, "REFERENTIAL_CONSTRAINTS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

93) Subclause 18.2.21, "CHECK_CONSTRAINTS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

94) Subclause 18.2.22, "KEY_COLUMN_USAGE view":

a) Conforming Entry SQL language shall not reference the Information Schema.

95) Subclause 18.2.23, "ASSERTIONS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

96) Subclause 18.2.24, "CHARACTER_SETS view":

a) Conforming Entry SQL language shall not reference the Information Schema.

97) Subclause 18.2.27, "VIEW_TABLE_USAGE view":

a) Conforming Entry SQL language shall not reference the Information Schema.

98) Subclause 18.2.28, "VIEW_COLUMN_USAGE view":

a) Conforming Entry SQL language shall not reference the Information Schema.

99) Subclause 18.2.29, "CONSTRAINT_TABLE_USAGE view":

a) Conforming Entry SQL language shall not reference the Information Schema.

100) Subclause 18.2.30, "CONSTRAINT_COLUMN_USAGE view":

a) Conforming Entry SQL language shall not reference the Information Schema.

101) Subclause 18.2.31, "COLUMN_DOMAIN_USAGE view":

a) Conforming Entry SQL language shall not reference the Information Schema.

102) Subclause 18.3.1, "SQL_LANGUAGES view":

a) Conforming Entry SQL language shall not reference the Information Schema.

Annex B
(Informative)

Implementation-defined elements

This Annex references those features that are identified in the body of this
ANSI American
ISO International
Standard as implementation-defined.

The term *implementation-defined* is used to identify characteristics that may differ between implementations, but that shall be defined for each particular implementation.

1) Subclause 4.2.1, "Character strings and collating sequences": The specific character set associated with the subtype of character string represented by the <key word>s NATIONAL CHARACTER is implementation-defined.

2) Subclause 4.6, "Numbers": Whether truncation or rounding is performed when trailing digits are removed from a numeric value is implementation-defined.

3) Subclause 4.6, "Numbers": When an approximation is obtained by truncation or rounding and there are more than one approximation, then it is implementation-defined which approximation is chosen.

4) Subclause 4.6, "Numbers": It is implementation-defined which numeric values have approximations obtained by rounding or truncation for a given approximate numeric type.

5) Subclause 4.6, "Numbers": The boundaries within which the normal rules of arithmetic apply are implementation-defined.

ANSI Only–SQL3

6) Subclause 4.14, "Row identifiers": It is implementation-defined whether base tables without supertables or subtables have row identifiers implicitly defined.

7) Subclause 4.14, "Row identifiers": The validity of row identifier mappings to host programs for durations beyond the SQL-transaction or open cursor, or simultaneously in different SQL-transactions, is implementation-defined.

8) Subclause 4.14, "Row identifiers": The source language data type to which row identifiers are mapped is implementation-defined.

9) Subclause 4.15, "Type conversions and mixing of data types": When converting between numeric data types, if least significant digits are lost, then it is implementation-defined whether rounding or truncation occurs.

10) Subclause 4.25, "Catalogs": The creation and destruction of catalogs is accomplished by implementation-defined means.

11) Subclause 4.25, "Catalogs": The set of catalogs that can be referenced in any SQL-statement, during any particular SQL-transaction, or during the course of an SQL-session is implementation-defined.

12) Subclause 4.25, "Catalogs": The default catalog for <module>s whose <module authorization clause> does not specify an explicit <catalog name> to qualify <schema name> is implementation-defined.

13) Subclause 4.28, "SQL-environment": The constituents of an SQL-environment beyond those specified in Subclause 4.28, "SQL-environment", are implementation-defined.

14) Subclause 4.28, "SQL-environment": The rules determining whether a <module> is within the environment are implementation-defined.

15) Subclause 4.29, "Modules": The mechanisms by which <module>s are created or destroyed are implementation-defined.

16) Subclause 4.29, "Modules": The manner in which an association between a <module> and an SQL-agent is defined is implementation-defined.

17) Subclause 4.29, "Modules": Whether a compilation unit may invoke or transfer control to other compilation units, written in the same or a different programming language is implementation-defined.

18) Subclause 4.32.1, "Status parameters": The negative (exception) values for the SQLCODE status parameter are implementation-defined.

19) Subclause 4.35, "Cursors": If a holdable-cursor that is not an INSENSITIVE cursor is held open for a subsequent SQL-transaction, then whether the effects of any changes made to SQL-data other than through that cursor (by this or any other SQL-transaction) will be visible through that cursor in the subsequent SQL-transacation before that cursor is closed is implementation-defined.

20) Subclause 4.39, "SQL-transactions": It is implementation-defined whether or not the execution of an SQL-data statement is permitted to occur within the same SQL-transaction as the execution of an SQL-schema statement. If it does occur, then the effect on any open cursor or deferred constraint is also implementation-defined.

21) Subclause 4.39, "SQL-transactions": If an implementation detects unrecoverable errors and implicitly initiates the execution of a <rollback statement>, an exception condition is raised with an *implementation-defined exception code*.

22) Subclause 4.40, "SQL-connections": It is implementation-defined how an implementation uses <SQL-server name> to determine the location, identity, and communication protocol required to access the SQL-server and initiate an SQL-session.

23) Subclause 4.41, "SQL-sessions": When an SQL-session is initiated other than through the use of an explicit <connect statement>, then an SQL-session associated with an implementation-defined SQL-server is initiated. The default SQL-server is implementation-defined.

24) Subclause 4.41, "SQL-sessions": The mechanism and rules by which an SQL-implementation determines whether a call to a <routine> is the last call within the last active <module> is implementation-defined.

25) Subclause 4.41, "SQL-sessions": An SQL-session uses one or more implementation-defined schemas that contain the instances of any global temporary tables, created local temporary tables, or declared local temporary tables within the SQL-session.

26) Subclause 4.41, "SQL-sessions": When an SQL-session is initiated, there is an implementation-defined default time zone used as the current default time zone displacement of the SQL-session.

27) Subclause 4.41, "SQL-sessions": When an SQL-session is initiated other than through the use of an explicit <connect statement>, there is an implementation-defined default <authorization identifier> that is used to for privilege checking for the execution of <SQL procedure statement>s contained in having an explicit <module authorization identifier>.

28) Subclause 5.1, "<SQL terminal character>": The end-of-line indicator (<newline>) is implementation-defined.

29) Subclause 5.3, "<literal>": The <character set name> character set used to represent national characters is implementation-defined.

30) Subclause 5.4, "Names and identifiers": If a <schema name> contained in a <schema name clause> but not contained in a <module> does not contain a <catalog name>, then an implementation-defined <catalog name> is implicit.

31) Subclause 5.4, "Names and identifiers": If a <schema name> contained in a <module authorization clause> does not contain a <catalog name>, then an implementation-defined <catalog name> is implicit.

32) Subclause 5.4, "Names and identifiers": Those <identifier>s that are valid <authorization identifier>s are implementation-defined.

33) Subclause 5.4, "Names and identifiers": Those <identifier>s that are valid <catalog name>s are implementation-defined.

34) Subclause 5.4, "Names and identifiers": All <form-of-use conversion name>s are implementation-defined.

35) Subclause 5.4, "Names and identifiers": The <entry name> of an entry point to a function defined as part of an abstract data type is implementation-defined.

36) Subclause 6.1, "<data type>": The <character set name> associated with NATIONAL CHARACTER is implementation-defined.

37) Subclause 6.1, "<data type>": If a <precision> is omitted, then an implementation-defined <precision> is implicit.

38) Subclause 6.1, "<data type>": The decimal precision of a data type defined as DECIMAL for each value specified by <precision> is implementation-defined.

39) Subclause 6.1, "<data type>": The precision of a data type defined as INTEGER is implementation-defined, but has the same radix as that for SMALLINT.

40) Subclause 6.1, "<data type>": The precision of a data type defined as SMALLINT is implementation-defined, but has the same radix as that for INTEGER.

41) Subclause 6.1, "<data type>": The binary precision of a data type defined as FLOAT for each value specified by <precision> is implementation-defined.

42) Subclause 6.1, "<data type>": The precision of a data type defined as REAL is implementation-defined.

43) Subclause 6.1, "<data type>": The precision of a data type defined as DOUBLE PRECISION is implementation-defined, but greater than that for REAL.

44) Subclause 6.1, "<data type>": For every <data type>, the limits of the <data type> are implementation-defined.

45) Subclause 6.1, "<data type>": The maximum lengths for character string types, variable-length character string types, bit string types, and variable-length bit string types are implementation-defined.

46) Subclause 6.1, "<data type>": If CHARACTER SET is not specified for <character string type>, then the character set is implementation-defined.

47) Subclause 6.1, "<data type>": The character set named SQL_TEXT is an implementation-defined character set that contains every character that is in <SQL language character> and all characters that are in other character sets supported by the implementation.

48) Subclause 6.1, "<data type>": For the <exact numeric type>s DECIMAL and NUMERIC, the maximum values of <precision> and of <scale> are implementation-defined.

49) Subclause 6.1, "<data type>": For the <approximate numeric type> FLOAT, the maximum value of <precision> is implementation-defined.

50) Subclause 6.1, "<data type>": For the <approximate numeric type>s FLOAT, REAL, and DOUBLE PRECISION, the maximum and minimum values of the exponent are implementation-defined.

51) Subclause 6.1, "<data type>": The maximum value of <time fractional seconds precision> is implementation-defined, but shall not be less than 6.

52) Subclause 6.1, "<data type>": The maximum values of <time precision> and <timestamp precision> for a <datetime type> are the same implementation-defined value.

53) Subclause 6.2, "<value specification> and <target specification>": Whether the character string of the <value specification>s CURRENT_USER, SESSION_USER, and SYSTEM_USER is variable-length or fixed-length, and its maximum lenth if it is variable-length or its length if it is fixed-length, are implementation-defined.

54) Subclause 6.2, "<value specification> and <target specification>": The value specified by SYSTEM_USER is an implementation-defined string that represents the operating system user who executed the <module> that contains the SQL-statement whose execution caused the SYSTEM_USER <general value specification> to be evaluated.

55) Subclause 6.8, "<set function specification>": The precision of the value derived from application of the COUNT function is implementation-defined.

56) Subclause 6.8, "<set function specification>": The precision of the value derived from application of the SUM function to a data type of exact numeric is implementation-defined.

57) Subclause 6.8, "<set function specification>": The precision and scale of the value derived from application of the AVG function to a data type of exact numeric is implementation-defined.

58) Subclause 6.8, "<set function specification>": The precision of the value derived from application of the SUM function or AVG function to a data type of approximate numeric is implementation-defined.

59) Subclause 6.9, "<numeric value function>": The precision of <position expression> is implementation-defined.

60) Subclause 6.9, "<numeric value function>": The precision of <extract expression> is implementation-defined. If <datetime field> specifies SECOND, then the scale is also implementation-defined.

61) Subclause 6.9, "<numeric value function>": The precision of <length expression> is implementation-defined.

62) Subclause 6.10, "<string value function>": The maximum length of <character translation> or <form-of-use conversion> is implementation-defined.

63) Subclause 6.14, "<cast specification>": Whether to round or truncate when casting to exact or approximate numeric data types is implementation-defined.

64) Subclause 6.17, "<numeric value expression>": When the data type of both operands of the addition. subtraction, multiplication, or division operator is exact numeric, the precision of the result is implementation-defined.

65) Subclause 6.17, "<numeric value expression>": When the data type of both operands of the division operator is exact numeric, the scale of the result is implementation-defined.

66) Subclause 6.17, "<numeric value expression>": When the data type of either operand of an arithmetic operator is approximate numeric, the precision of the result is implementation-defined.

67) Subclause 6.17, "<numeric value expression>": Whether to round or truncate when performing division is implementation-defined.

68) Subclause 6.21, "<interval value expression>": When an interval is produced from the difference of two datetimes, the choice of whether to round or truncate is implementation-defined.

69) Subclause 7.15, "<recursive union>": The implicit limit value in a <limit clause> is an implementation-defined positive integer.

70) Subclause 7.15, "<recursive union>": The order of rows returned when PREORDER is specified is implementation-defined.

71) Subclause 9.1, "Retrieval assignment": If a value *V* is approximate numeric and a target *T* is exact numeric, then whether the approximation of *V* retrieved into *T* is obtained by rounding or by truncation is implementation-defined.

72) Subclause 9.1, "Retrieval assignment": If a value *V* is interval with a greater precision than a target *T*, then it is implementation-defined whether the appproximation of *V* retrieved into *T* is obtained by rounding or by truncation.

73) Subclause 9.2, "Store assignment": If a value *V* is approximate numeric and a target *T* is exact numeric, then whether the approximation of *V* stored into *T* is obtained by rounding or by truncation is implementation-defined.

74) Subclause 9.2, "Store assignment": If a value *V* is interval with a greater precision than a target *T*, then it is implementation-defined whether the appproximation of *V* stored into *T* is obtained by rounding or by truncation.

75) Subclause 9.3, "Set operation result data types and nullabilities": If all of the data types in *DTS* are exact numeric, then the result data type is exact numeric with implementation-defined precision.

76) Subclause 9.3, "Set operation result data types and nullabilities": If any data type in *DTS* is approximate numeric, then each data type in *DTS* shall be numeric and the result data type is approximate numeric with implementation-defined precision.

77) Subclause 10.1, "<interval qualifier>": The maximum value of <interval leading field precision> is implementation-defined, but shall not be less than 2.

78) Subclause 10.1, "<interval qualifier>": The maximum value of <interval fractional seconds precision> is implementation-defined, but shall not be less than 2.

- 2 list elements moved to Part 4

79) Subclause 10.5, "<character set specification>": The <standard character repertoire name>s, <implementation-defined character repertoire name>s, <standard universal character form-of-use name>s, and <implementation-defined universal character form-of-use name>s that are supported are implementation-defined.

80) Subclause 11.1, "<schema definition>": If <schema character set specification> is not specified, then a <schema character set specification> containing an implementation-defined <character set specification> is implicit.

81) Subclause 11.1, "<schema definition>": If <schema path specification> is not specified, then a <schema path specification> containing an implementation-defined <schema name list> is implicit.

82) Subclause 11.1, "<schema definition>": If AUTHORIZATION <authorization identifier> is not specified, then an <authorization identifier> equal to the implementation-defined <authorization identifier> for the SQL-session is implicit.

83) Subclause 11.1, "<schema definition>": The privileges necessary to execute the <schema definition> are implementation-defined.

ANSI Only–SQL3

84) It is implementation-defined whether a row identifier for a table is implicitly defined in the absence of subtables or supertables.

85) Subclause 11.53, "<operators definition>": The Access Rules are implementation-defined.

86) Subclause 11.55, "<role definition>": The Access Rules are implementation-defined.

87) Subclause 11.39, "<collation definition>": The <standard collation name>s and <implementation-defined collation name>s that are supported are implementation-defined.

88) Subclause 11.39, "<collation definition>": The collating sequence resulting from the specification of EXTERNAL in a <collation definition> may be implementation-defined.

89) Subclause 11.41, "<translation definition>": The <standard translation name>s and <implementation-defined translation name>s that are supported are implementation-defined.

90) Subclause 11.59, "<revoke statement>": When loss of the USAGE privilege on a character set causes a module to be determined to be a lost module, the impact on that module is implementation-defined.

91) Subclause 12.2, "<module>": If the explicit or implicit <schema name> does not specify a <catalog name>, then an implementation-defined <catalog name> is implicit.

92) Subclause 12.2, "<module>": If <module path specification> is not specified, then a <module path specification> containing an implementation-defined <schema name list> is implicit.

93) Subclause 12.2, "<module>": If a <module character set specification> is not specified, then a <module character set specification> that specifies the implementation-defined character set that contains every character that is in <SQL language character> is implicit.

- Several list elements moved to Part 4

94) Subclause 13.1, "<declare cursor>": Whether null values shall be considered greater than or less than all non-null values in determining the order of rows in a table associated with a <declare cursor> is implementation-defined.

95) Subclause 13.1, "<declare cursor>": Whether an implementation is able to disallow significant changes that would not be visible through a currently open cursor is implementation-defined.

96) Subclause 13.2, "<open statement>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

97) Subclause 13.6, "<delete statement: positioned>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

98) Subclause 13.7, "<delete statement: searched>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

99) Subclause 13.8, "<insert statement>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

100) Subclause 13.9, "<update statement: positioned>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

101) Subclause 13.10, "<update statement: searched>": The extent to which an implementation may disallow independent changes that are not significant is implementation-defined.

102) Subclause 14.2, "<set transaction statement>": The isolation level that is set for a transaction is an implementation-defined isolation level that will not exhibit any of the phenomena that the explicit or implicit <level of isolation> would not exhibit, as specified in Table 10, "SQL-transaction isolation levels and the three phenomena".

103) Subclause 14.5, "<savepoint statement>": The maximum number of savepoints per SQL-transaction is implementation-defined.

104) Subclause 14.8, "<rollback statement>": The status of any open cursors in any <module> associated with the current SQL-transaction that were opened by that SQL-transaction before the establishment of a savepoint to which a rollback is executed is implementation-defined.

105) Subclause 15.1, "<connect statement>": If <user name> is not specified, then an implementation-defined <user name> for the SQL-connection is implicit.

106) Subclause 15.1, "<connect statement>": The restrictions on whether or not the <user name> must be identical to the <module authorization identifier> for the <module> that contains the <routine> that contains the <connect statement> are implementation-defined.

107) Subclause 15.1, "<connect statement>": If DEFAULT is specified, then the method by which the default SQL-server is determined is implementation-defined.

108) Subclause 15.1, "<connect statement>": The method by which <SQL-server name> is used to determine the appropriate SQL-server is implementation-defined.

109) Subclause 16.2, "<set session authorization identifier statement>": Whether or not the <authorization identifier> for the SQL-session can be set to an <authorization identifier> other than the <authorization identifier> of the SQL-session when the SQL-session is started is implementation-defined, as are any restrictions pertaining to such changes.

110) Subclause 17.1, "<get diagnostics statement>": The actual length of variable-length character items in the diagnostics area is implementation-defined but not less than 128.

111) Subclause 17.1, "<get diagnostics statement>": The character string value set for CLASS_ORIGIN and SUBCLASS_ORIGIN for an implementation-defined class code or subclass code is implementation-defined, but shall not be 'ISO 9075'.

112) Subclause 17.1, "<get diagnostics statement>": The value of MESSAGE_TEXT is an implementation-defined character string.

113) Subclause 18.4.15, "TABLE_CONSTRAINTS base table": If the containing <add table constraint definition> does not specify a <constraint name definition>, then the value of CONSTRAINT_NAME is implementation-defined.

114) Subclause 18.4.29, "CHARACTER_SETS base table": The values of FORM_OF_USE and NUMBER_OF_CHARACTERS, in the row for the character set INFORMATION_SCHEMA.SQL_TEXT, are implementation-defined.

115) Subclause 18.4.30, "COLLATIONS base table": The value of COLLATION_DEFINITION is implementation-defined.

116) Subclause 18.4.35, "SQL_LANGUAGES base table": The value of SQL_LANGUAGE_IMPLEMENTATION is implementation-defined. If the value of SQL_LANGUAGE_SOURCE is not 'ISO 9075', then the value of all other columns is implementation-defined.

117) Subclause 19.1, "SQLSTATE": The character set associated with the class value and subclass value of the SQLSTATE parameter is implementation-defined.

118) Subclause 19.1, "SQLSTATE": The values and meanings for classes and subclasses that begin with one of the <digit>s '5', '6', '7', '8', or '9' or one of the <simple Latin upper case letter>s 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', or 'Z' are implementation-defined. The values and meanings for all subclasses that are associated with implementation-defined class values are implementation-defined.

119) Subclause 19.2, "SQLCODE": The negative values returned in an SQLCODE parameter to indicate exception conditions are implementation-defined.

120) Clause 20, "Conformance": The method of flagging nonconforming SQL language or processing of conforming SQL language is implementation-defined, as is the list of additional <key word>s that may be required by the implementation.

Annex C
(Informative)

Implementation-dependent elements

This Annex references those places where this
ANSI American
ISO International
Standard states explicitly that the actions of a conforming implementation are implementation-dependent.

The term *implementation-dependent* is used to identify characteristics that may differ between implementations, but that are not necessarily specified for any particular implementation.

1) Subclause 3.3.4, "Use of terms": The effect on <target specification>s and SQL descriptor areas of an SQL-statement that terminates with an exception condition, unless explicitly defined by this
ANSI American
ISO International
standard, is implementation-dependent.

2) Subclause 3.3.4, "Use of terms": If more than one condition could have occurred as a result of execution of a statement, then it is implementation-dependent whether diagnostic information pertaining to more than one condition is made available.

3) Subclause 3.3.4, "Use of terms": The treatment of language that does not conform to the SQL Formats and Syntax Rules is implementation-dependent.

4) Subclause 3.3.4, "Use of terms": It is implementation-dependent whether expressions are actually evaluated left-to-right when the precedence is not otherwise determined by the Formats or by parentheses.

5) Subclause 3.3.4, "Use of terms": If evaluation of the inessential parts of an expression or search condition would cause an exception condition to be raised, then it is implementation-dependent whether or not that condition is raised.

6) Subclause 3.3.3, "Specification of the Information Schema" The actual objects on which the Information Schema views are based are implementation-dependent.

7) Subclause 4.1, "Data types": The physical representation of a value of a data type is implementation-dependent.

8) Subclause 4.1, "Data types": The null value or values for each data type is implementation-dependent.

9) Subclause 4.18, "Nulls": A null state is a classified null value that is implementation-dependent but distinct from the both the general null value and all other classified null values for the domain or column.

10) Subclause 4.20, "Tables": Because global temporary table contents are distinct within SQL-sessions, and created local temporary tables are distinct within <module>s within SQL-sessions, the effective <schema name> of the schema in which the global temporary table or the created local temporary table is instantiated is an implementation-dependent <schema name> that may be thought of as having been effectively derived from the <schema name> of the schema in which the global temporary table or created local temporary table is defined and the implementation-dependent SQL-session identifier associated with the SQL-session.

11) Subclause 4.20, "Tables": The effective <schema name> of the schema in which the created local temporary table is instantiated may be thought of as being further qualified by a unique implementation-dependent name associated with the <module> in which the created local temporary table is referenced.

12) Subclause 4.20, "Tables": Whether a temporary viewed table is materialized is implementation-dependent.

13) Subclause 4.33, "Diagnostics area": The actual size of the diagnostics area is implementation-dependent when the SQL-agent does not specify the size.

14) Subclause 4.33, "Diagnostics area": The ordering of the the information about conditions placed into the diagnostics area is implementation-dependent, except that the first condition in the diagnostics area always corresponds to the condition corresponding to the SQLSTATE or SQLCODE value.

15) Subclause 4.35, "Cursors": If the <declare cursor> does not include an <order by clause>, or includes an <order by clause> that does not specify the order of the rows completely, then the rows of the table have an order that is defined only to the extent that the <order by clause> specifies an order and is otherwise implementation-dependent.

16) Subclause 4.35, "Cursors": When the ordering of a cursor is not defined by an <order by clause>, the relative position of two rows is implementation-dependent.

17) Subclause 4.35, "Cursors": The effect on the position and state of an open cursor when an error occurs during the execution of an SQL-statement that identifies the cursor is implementation-dependent.

18) Subclause 4.35, "Cursors": If a cursor is open and a change is made to SQL-data from within the same SQL-transaction other than through that cursor, and the cursor was not declared SENSITIVE or INSENSITIVE, then whether that change will be visible through that cursor before it is closed is implementation-dependent.

19) Subclause 4.37, "Privileges and roles": The mapping of <authorization identifier>s to operating system users is implementation-dependent.

20) Subclause 4.37, "Privileges and roles": When an SQL-session is initiated, the current <authorization identifier> for the SQL-session is determined in an implementation-dependent manner, unless the session is initiated using a <connect statement>.

21) Subclause 4.38, "SQL-agents": An SQL-agent is an implementation-dependent entity that causes the execution of SQL-statements.

22) Subclause 4.39, "SQL-transactions": The schema definitions that are implicitly read on behalf of executing an SQL-statement are implementation-dependent.

23) Subclause 4.41, "SQL-sessions": A unique implementation-dependent SQL-session identifier is associated with each SQL-session.

24) Subclause 4.42, "Client-server operation": The <module name> of the <module> that is effectively materialized on an SQL-server is implementation-dependent.

25) Subclause 4.42, "Client-server operation": Diagnostic information is passed to the diagnostics area in the SQL-client is passed in an implementation-dependent manner.

26) Subclause 4.42, "Client-server operation": The effect on diagnostic information of incompatibilities between the character repertoires supported by the SQL-client and SQL-server environments is implementation-dependent.

27) Subclause 5.3, "<literal>": The method of representing an object identifier as an <oid value> is implementation-dependent.

28) Subclause 5.3, "<literal>": Within the definition of an <oid literal>, the allowed <oid value>s are implementation-dependent.

29) Subclause 6.3, "<item reference>": The implicit qualifier of a column reference for which there is more than one possible qualifier with most local scope is implementation-dependent.

30) Subclause 6.11, "<datetime value function>": The time of evaluation of the CURRENT_DATE, CURRENT_TIME, and CURRENT_TIMESTAMP functions during the execution of an SQL-statement is implementation-dependent.

31) Subclause 6.21, "<interval value expression>": The start datetime used for converting intervals to scalars for subtraction purposes is implementation-dependent.

32) Subclause 7.1, "<row value constructor>": The names of the columns of a <row value constructor> that specifies a <row value constructor list> are implementation-dependent.

33) Subclause 7.13, "<query specification>": When a column is not named by an <as clause> and is not derived from a single column reference, then the name of the column is implementation-dependent.

34) Subclause 7.13, "<query specification>": If a <simple table> is neither a <query specification> nor an <explicit table>, then the name of each column of the <simple table> is implementation-dependent.

35) Subclause 7.13, "<query specification>": If a <non-join query term> is not a <non-join query primary> and the <column name> of the corresponding columns of both tables participating in the <non-join query term> are not the same, then the result column has an implementation-dependent <column name>.

36) Subclause 7.13, "<query specification>": If a <non-join query expression> is not a <non-join query term> and the <column name> of the corresponding columns of both tables participating in the <non-join query expression > are not the same, then the result column has an implementation-dependent <column name>.

37) Subclause 8.2, "<comparison predicate>": When the operations MAX, MIN, DISTINCT, and references to a grouping column refer to a variable-length character string or a variable-length bit string, the specific value selected from the set of equal values is implementation-dependent.

38) Subclause 9.3, "Set operation result data types and nullabilities": The specific character set chosen for the result is implementation-dependent, but shall be the character set of one of the data types in *DTS*.

- 3 list elements moved to Part 4

39) Subclause 11.5, "<table definition>": The <abstract data type name> of an abstract data type specified in a <table definition> without specifying "OF NEW TYPE *ADTN*" is implementation-dependent.

40) Subclause 11.10, "<table constraint definition>": The <constraint name> of a constraint that does not specify a <constraint name definition> is implementation-dependent.

41) Subclause 11.12, "<referential constraint definition>": The specific value to use for cascading among various values that are not distinct is implementation-dependent.

42) Subclause 11.28, "<domain definition>": The <constraint name> of a constraint that does not specify a <constraint name definition> is implementation-dependent.

43) Subclause 11.39, "<collation definition>": The collation of characters for which a collation is not otherwise specified is implementation-dependent.

- Several list elements moved to Part 4

44) Subclause 13.1, "<declare cursor>": If a <declare cursor> does not contain an
ANSI <order by clause> and the table is not an ordered table,
ISO <order by clause>,
then the ordering of rows in the table associated with that <declare cursor> is implementation-dependent.

45) Subclause 13.1, "<declare cursor>": If a <declare cursor> contains an <order by clause> and a group of two or more rows in the table associated with that <declare cursor> contain values that

Case:

a) are the same null value, or

b) compare equal according to Subclause 8.2, "<comparison predicate>" in all columns specified in the <order by clause>, then the order in which rows in that group are returned is implementation-dependent.

46) Subclause 13.3, "<fetch statement>": The order of assignment to targets in the <fetch target list> of values returned by a <fetch statement>, other than status parameters, is implementation-dependent.

47) Subclause 13.3, "<fetch statement>": If an error occurs during assignment of a value to a target during the execution of a <select statement: single row>, then the values of targets other than status parameters are implementation-dependent.

48) Subclause 13.3, "<fetch statement>": If an exception condition occurs during the assignment of a value to a target, then the values of all targets are implementation-dependent and *CR* remains positioned on the current row.

49) Subclause 13.5, "<select statement: single row>": The order of assignment to targets in the <select target list> of values returned by a <select statement: single row>, other than status parameters, is implementation-dependent.

50) Subclause 13.5, "<select statement: single row>": If the cardinality of the <query expression> is greater than 1, then it is implementation-dependent whether or not values are assigned to the targets identified by the <select target list>.

51) Subclause 13.5, "<select statement: single row>": If an error occurs during assignment of a value to a target during the execution of a <select statement: single row>, then the values of targets other than status parameters are implementation-dependent.

52) Subclause 13.13, "<temporary view declaration>": Whether a temporary viewed table is materialized is implementation-dependent.

53) Subclause 14.2, "<set transaction statement>": If <number of conditions> is not specified, then an implementation-dependent value not less than 1 is implicit.

54) Subclause 14.5, "<savepoint statement>": If <savepoint specifier> is specified as <simple target specification>, then *S* is set to an implementation-dependent value that is non-0 and different from all other values that have been used to identify savepoints within the current SQL-transaction.

55) Subclause 15.3, "<disconnect statement>": If ALL is specified, then *L* is a list representing every active SQL-connection that has been established by a <connect statement> by the current SQL-agent and that has not yet been disconnected by a <disconnect statement>, in an implementation-dependent order.

56) Subclause 17.1, "<get diagnostics statement>": The value of ROW_COUNT following the execution of an SQL-statement that does not directly result in the execution of a <delete statement: searched>, an <insert statement>, or an <update statement: searched> is implementation-dependent.

57) Subclause 17.1, "<get diagnostics statement>": If <condition number> has a value other than 1, then the association between <condition number> values and specific conditions raised during evaluation of the General Rules for that SQL-statement is implementation-dependent.

58) Subclause 18.2, "Information Schema": The actual <query specification>s required to create the Information Schema tables are implementation-dependent.

59) Subclause 18.4.29, "CHARACTER_SETS base table": The value of DEFAULT_COLLATE_NAME for default collations specifying the order of characters in a repertoire is implementation-dependent.

Annex D
(Informative)

Deprecated features

It is intended that the following features will be removed at a later date from a revised version of this
ANSI American
ISO International
Standard:

1) SQLCODE

2) The <unsigned integer> option of <sort specification> of <declare cursor>.

3) The omission of <semicolon> in <module contents>.

Annex E
(Informative)

Incompatibilities with X3.135-1992 and ISO/IEC 9075:1992

This
ANSI American
ISO International
Standard introduces some incompatibilities with the earlier version of Database Language SQL as specified in
ANSI ANS X3.135-1992.
ISO ISO/IEC 9075:1992.
Unless specified in this Annex, features and capabilities of Database Language SQL are compatible with the earlier version of this
ANSI American
ISO International
Standard.

1) In
ANSI ANSI X3.135-1992,
ISO ISO/IEC 9075:1992,
Subclause 12.3, “<procedure>”, a <parameter declaration list> had an alternative “<parameter declaration> . . . ” (that is, a parameter list not surrounded by parentheses and with the individual component parameter declarations not separated by commas). This option was listed in
ANSI ANSI X3.135-1992,
ISO ISO/IEC 9075:1992,
as a deprecated feature. In
ANSI ANSI X3.135-199*x*,
ISO ISO/IEC 9075:199*x*,
the equivalent Subclause 11.3, "<routine>", in Part 4 of this
ANSI American
ISO International
Standard, does not contain this option. Modules that used this deprecated feature may be converted to conforming SQL by inserting a comma between each pair of <parameter declaration>s and placing a left parenthesis before and a right parenthesis after the entire parameter list.

2) A number of additional <reserved word>s have been added to the language. These <reserved word>s are:

— ACTION

— ACTOR

— AFTER

— ALIAS

— ASYNC

ANSI Only—caused by ISO changes not yet considered by ANSI

— ATTRIBUTES

— BEFORE

— BINARY

— BLOB

— BOOLEAN

— BREADTH

— CLOB

— COMPLETION

— CURRENT_PATH

— CYCLE

— DATA

— DEPTH

— DESTROY

— DICTIONARY

— EACH

ANSI Only–SQL3

— ELEMENT

— ELSEIF

— EQUALS

— FACTOR

— FREE

— GENERAL

— HOLD

— HOST

— IGNORE

— INSTEAD

— LESS

— LARGE

— LIMIT

ANSI Only–SQL3

— LIST

— LOCATOR

— MODIFY

— NCLOB

— NEW

ANSI Only–SQL3

— NEW_TABLE

— NO

— NONE

— OBJECT

— OFF

— OID

— OLD

ANSI Only–SQL3

— OLD_TABLE

— OPERATION

— OPERATOR

— OPERATORS

— OVERLAY

— PARAMETERS

— PATH

— PENDANT

— POSTFIX

— PREFIX

— PREORDER

— PRIVATE

— PROTECTED

— RECURSIVE

— REFERENCING

— REPLACE

— ROLE

— ROUTINE

— ROW

— SAVEPOINT

— SEARCH

— SENSITIVE

— SEQUENCE

— SESSION

— SIMILAR

— SPACE

— SQLEXCEPTION

— SQLWARNING

— START

— STATE

— STRUCTURE

— SYMBOL

— TERM

— TEST

ANSI Only—caused by ISO changes not yet considered by ANSI

— THERE

— TREAT

— TRIGGER

— TYPE

— UNDER

— UPDATABLE

— VARIABLE

- 1 list entry deleted

— VISIBLE

— WAIT

— WITHOUT

Annex F
(Informative)

Maintenance and interpretation of SQL

ANSI Only–SQL3

ANSI Accredited Committee X3 provides formal procedures for revision, maintenance, and interpretation of ANSI Standards produced by X3. Section 5.2.3 of the Organization, Rules, and Procedures of X3 (X3/SD-2), "Maintenance of American National Standards", specifies procedures for defect management of ANSI X3 standards, including Errata, Amendments, and Interpretations. Errata and Interpretations are published by X3 in an SQL Information Bulletin. Amendments are processed under procedures that guarantee adequate public review before adoption.

Since publication of ANSI X3.135-1992, the following items have resulted in formal interpretations of Database Language SQL.

1) *to be provided*

The original questions and an explanation of the X3 interpretations, and correction of a number of SQL Errata, can be found in SQL Information Bulletin #*to be provided* (SQLIB-*to be provided*) published April 19, 1991, by Global Engineering Documents, Inc.

Since publication of ANSI X3.135-19892, several new defects have been discovered in the SQL language, leading to creation of the following defect reports against the equivalent International SQL standard, ISO/IEC 9075:1992. Numbers in parentheses refer to ISO/IEC JTC1/SC21/WG3 documents that identify the defects.

1) *to be provided*

The SQL language corrections proposed in each defect report were accepted by SC21/WG3 in *date to be provided* (see SC21 N*to be provided*, *city to be provided* WG3 Resolutions). These clarifications have all been endorsed by X3 technical committee X3H2. Formal processing within ANSI and further processing within ISO/IEC has been superseded by adoption of ISO/IEC 9075:199*x* as a replacement standard for ISO/IEC 9075:1992 and by adoption of ANSI X3.135-199*x* as a replacement standard for ANSI X3.135-1992.

All corrections to SQL proposed by these defect reports are included in this American National Standard.

Potential new questions or new defect reports addressing the specifications of this American National Standard should be communicated to:

X3 Secretariat, Computer and Business Equipment Manufacturers Association (CBEMA)
1250 Eye St., Suite 200
Washington, DC 20005.

ISO Only–SQL3

ISO/IEC JTC1 provides formal procedures for revision, maintenance, and interpretation of JTC1 Standards. Section 6.13 of the JTC1 Directives, "Maintenance/correction of defects in JTC1 Standards", specifies procedures for creating and processing "defect reports". Defect reports may result in technical corrigenda, amendments, interpretations, or other commentary on an existing International Standard. In addition, SC21, the JTC1 subcommittee that developed this International Standard, provides procedures for raising new "questions" about topics related to existing SC21 projects. Questions may result in interpretations, new project proposals, or possibly new defect reports.

Since publication of ISO/IEC 9075:1992, the following SC21 questions have resulted in formal interpretations of Database Language SQL. The first number in parentheses identifies the SC21 document in which the question was first raised, and the second number identifies the SC21 document in which the proposed interpretation was formally adopted.

1) *to be provided*

Since publication of ISO/IEC 9075:1992, several new defects have been discovered in the SQL language, leading to creation of the following defect reports.

1) *to be provided*

The SQL language corrections proposed in each defect report were accepted by SC21/WG3 in *date to be provided* (see SC21 N*to be provided*, *city to be provided* WG3 Resolutions). Further processing within SC21 was superseded by adoption of ISO/IEC 9075:199*x* as a replacement standard for ISO/IEC 9075:1992. All corrections to SQL proposed by these defect reports are included in this International Standard.

Potential new questions or new defect reports addressing the specifications of this International Standard should be communicated to:

Secretariat, ISO/IEC JTC1/SC21/WG3
Standards Council of Canada
Ottawa, Ontario
Canada.

Annex G
(Informative)

Standard type templates and types

****Editor's Note****
This Annex has not yet been updated to reflect the introduction of the non-instantiable collection supertype and adjusted collection type hierarchy.

G.1 Introduction

Standard type templates and types are defined in this Clause by using the structure of <type template definition> and <abstract data type definition> to describe their interfaces. The semantics of their public <routine declaration>s and <column definition>s will generally be provided informally.

These standard type templates and types are informally grouped into interrelated families, with the type template and type names of a family all sharing a common prefix ending with an underscore.

Within the Subclause defining such a family, any concepts and terminology specific to that family will be included, for example where the concepts and terminology correspond to those of some other standard.

G.2 SQL type templates and types

The type templates and types in this family are designed to correspond to the functionality of parts of SQL itself.

The SQL_Table template defines the functions SQL_Insert, SQL_Select, SQL_Update, and SQL_Delete to correspond to the data manipulation language statements INSERT, SELECT, UPDATE, and DELETE. Other functions correspond to predicates and operators on tables.

The SQL_Set template inherits the functionality of an SQL_Table, and imposes the constraint that a set cannot contain duplicate elements. It also defines a Subset function.

The SQL_List template also inherits the functionality of an SQL_Table, refines the semantics of some inherited functions, and introduces additional functions Element and Position related to the ordering of elements in the list.

The SQL_Empty_Table, SQL_Empty_Set, and SQL_Empty_list types define tables, sets, and lists that are always empty, and are thus not constrained by any element type.

G.2.1 Definitions used in this Subclause

To Be Supplied

G.2.2 Concepts used in this Subclause

To Be Supplied

ANSI Only—caused by ISO changes not yet considered by ANSI

G.2.3 SQL_Table type template

Function

SQL_Table provides functions to carry out insertion, selection, updating, and deletion on SQL tables.

Definition

```
CREATE TYPE TEMPLATE  SQL_Table ( :Element_Type TYPE )

           !! default constructor and destructor signatures are to be
           !! inserted here when the naming problem is resolved

       (

       CAST ( :GEN_TYPE AS :Element_Type WITH Table_to_Element ) ; ,

       CAST ( :GEN_TYPE AS SQL_Set ( :Element_Type ) WITH Distinct ) ; ,

       CAST ( :GEN_TYPE AS SQL_List ( :Element_Type ) WITH Table_to_List ) ; ,

       CAST ( SQL_Empty_Table AS :GENTYPE WITH Empty_Table_to_Table ) ; ,

       FUNCTION SQL_Insert ( :tabref :GEN_TYPE ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION SQL_Select ( :tabref :GEN_TYPE ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) )
                RETURNS SQL_Table(Object) ; ,  !! assuming Object is a supertype of all
       ADT

       FUNCTION SQL_Update ( :tabref :GEN_TYPE ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION SQL_Delete ( :tabref :GEN_TYPE ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION In ( :table :GEN_TYPE, :elem :Element_Type )
                RETURNS BOOLEAN ; ,

       FUNCTION Exists ( :table :GEN_TYPE ) RETURNS BOOLEAN ; ,

       FUNCTION Unique ( :table :GEN_TYPE ) RETURNS BOOLEAN ; ,

       FUNCTION For_Some ( :table :GEN_TYPE,
                           :pred CHARACTER VARYING (max_pred_length) ,
                           :descr_name CHARACTER VARYING (max_name_length) )
                RETURNS BOOLEAN ; ,
```

```
FUNCTION For_All ( :table :GEN_TYPE ,
                   :pred CHARACTER VARYING (max_pred_length) ,
                   :descr_name CHARACTER VARYING (max_name_length) )
         RETURNS BOOLEAN ; ,

FUNCTION Average ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Maximum ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Minimum ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Sum ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Count ( :table :GEN_TYPE ) RETURNS count_type ; ,

FUNCTION Distinct ( :table :GEN_TYPE ) RETURNS SQL_Set ( :Element_Type ) ; ,

FUNCTION Union ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Union_All ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Except ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Except_All ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Intersect ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

FUNCTION Intersect_All ( :table :GEN_TYPE, :table2 :GEN_TYPE )
         RETURNS :GEN_TYPE ; ,

PRIVATE:

FUNCTION Table_to_Element ( :table :GEN_TYPE ) RETURNS :Element_Type ; ,

FUNCTION Table_to_List ( :table :GEN_TYPE )
         RETURNS SQL_List ( :Element_Type ) ; ,

FUNCTION Empty_Table_to_Table ( :target :GEN_TYPE ) RETURNS :GEN_TYPE ;

) ;
```

Note: I don't know how to fix the Definition, which has a reference to :descr_name>. Perhaps some part of this needs to be moved to Part *Z*, SQL/Bindings.

Description

1) Let *T* be any abstract or predefined data type. An instance of a generated type SQL_Table(*T*) is a table, each of whose rows is of type *T*.

2) The values of *max_stmt_length*, *max_name_length*, and *max_pred_length* are implementation-dependent maximum lengths for an <SQL procedure statement>, a <descriptor name>, and a <predicate> respectively. *count_type* is the data type of the result of a <set function specification> that simply contains COUNT.

Note: I don't know how to fix this Rule, which refers to a <descriptor name>. Perhaps some part of it should be moved to Part *Z*, SQL/Bindings. Without knowing how to deal with this, I have simply copied the entire Description to Part *Z*.

3) Let *F* be an SQL_Insert, SQL_Select, SQL_Update, or SQL_Delete function. Let *T*, *S*, and *D* be the contents of the arguments passed to the *tabref*, *stmt*, and *descr_name* parameters respectively of *F*.

4) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation in SQL_Table function*.

   a) Case:

      i) If *F* is an SQL_Insert function and either *S* does not conform to the Format, Syntax Rules, and Access Rules of an <insert statement>, or *S* is an <insert statement> whose immediately contained <table reference> does not identify the same base table as *T*.

      ii) If *F* is an SQL_Select function and either *S* does not conform to the Format, Syntax Rules, and Access Rules of a <query specification>, or *S* is a <query specification> simply containing a <from clause> whose first immediately contained <table reference> does not identify the same base table as *T*.

      iii) If *F* is an SQL_Update function and either *S* does not conform to the Format, Syntax Rules, and Access Rules of an <update statement: searched>, or *S* is an <update statement: searched> whose immediately contained <table reference> does not identify the same base table as *T*.

      iv) If *F* is an SQL_Delete function and either *S* does not conform to the Format, Syntax Rules, and Access Rules of a <delete statement: searched>, or *S* is a <delete statement: searched> whose immediately contained <table reference> does not refer to the same base table as *T*.

   b) *S* contains a parameter reference or an SQL variable reference.

5) When *F* is invoked, all General Rules of the corresponding SQL construct apply to *S*.

6) If *F* is an SQL_Select function, then the result *TQS* of the <query specification> *S* is returned as the result of invocation of *F*.

7) The In function is defined by the Syntax Rules, Access Rules, and General Rules of <in predicate>. The arguments :table and :elem correspond to the evaluated <table subquery> and the <row value expression> of the <in predicate>, respectively.

8) The Unique function is defined by the Syntax Rules, Access Rules, and General Rules of <unique predicate>. The argument :table corresponds to the evaluated <table subquery> of the <unique predicate>.

9) The Exists function is defined by the Syntax Rules, Access Rules, and General Rules of <exists predicate>. The argument :table corresponds to the evaluated <table subquery> of the <exists predicate>.

10) The For_Some and For_All functions are defined by the Syntax Rules, Access Rules, and General Rules of a <quantified predicate> immediately containing an <existential clause> *EC* or a <universal clause> *UC* respectively. The argument :table corresponds to the first <table reference> of the <table reference list> immediately contained in *EC* or *UC*.

11) The Average, Maximum, Minimum, Sum, and Count functions are defined by the Syntax Rules, Access Rules, and General Rules of the <set function specification>. The argument :table corresponds to the table *TXA* of the <set function specification>.

12) The Distinct function is defined by the Syntax Rules, Access Rules, and General Rules of DISTINCT in a <query specification>.

13) The Union, Union_All, Except, Except_All, Intersect, and Intersect_All functions are defined by the Syntax Rules, Access Rules, and General Rules of the corresponding operators of a <query expression>.

14) If an SQL_Table has only a single element, then the Table_to_Element function returns that element; otherwise an exception condition is raised: *data exception—not singleton collection in cast*.

15) The Table_to_List function returns an SQL_List containing the same elements as the SQL_Table, in an implementation-dependent order.

16) The Empty_Table_to_Table function returns a table of data type SQL_Table(*T*) that is empty, *i.e.*, contains no rows.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no data type generators.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only—caused by ISO changes not yet considered by ANSI

G.2.4 SQL_Set type template

Function

SQL_Set provides the ability to create and manipulate sets of elements of a specified type.

The SQL_Set type template inherits the functionality of SQL_Table, and applies the constraint that an SQL_Set cannot contain duplicate members. It also defines a subset predicate function.

Definition

```
CREATE TYPE TEMPLATE SQL_Set ( :Element_Type TYPE )
                   UNDER SQL_Table ( :Element_Type )

       (
           !! default constructor and destructor signatures are to be
           !! inserted here when naming problem is resolved

       CAST ( :GEN_TYPE AS :Element_Type WITH Table_to_Element ) ; ,

       CAST ( :GEN_TYPE AS SQL_Table ( :Element_Type ) WITH Set_to_Table ) ; ,

       CAST ( :GEN_TYPE AS SQL_List ( :Element_Type ) WITH Table_to_List ) ; ,

       CAST ( SQL_Empty_Set AS :GENTYPE WITH Empty_Set_to_Set ) ; ,

       CHECK(Unique(VALUE)) ,

       FUNCTION Subset ( :set :GEN_TYPE, :set2 :GEN_TYPE )
                RETURNS BOOLEAN ;

       PRIVATE:

       FUNCTION Set_to_Table ( :set :GEN_TYPE )
                RETURNS SQL_Table ( :Element_Type ) ; ,

       FUNCTION Empty_Set_to_Set ( :target :GEN_TYPE ) RETURNS :GEN_TYPE ;

       ) ;
```

Description

1) Let *T* be any data type. An instance of a generated type SQL_Set(*T*) is a table, each of whose rows is of type *T*.

2) The CHECK condition applies the Unique function, inherited from SQL_Table, to the value of an instance of an SQL_Set (*T*) for some type *T*. This checks for uniqueness of the elements of type *T* of the SQL_Set(*T*).

3) The Subset function returns true if, for every element *E* of the SQL_Set identified by :set, In(:set2, *E*) is true; otherwise it returns false.

4) The Set_to_Table function returns an SQL_Table containing the same elements as the SQL_Set.

5) The Empty_Set_to_Set function returns a set of data type SQL_Set(*T*) that is empty, *i.e.*, contains no rows.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no data type generators.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only—caused by ISO changes not yet considered by ANSI

G.2.5 SQL_List type template

Function

SQL_List provides the ability to create and manipulate lists of elements of a specified type.

The SQL_List type template inherits the functionality of SQL_Table (redefining several of the functions to take account of ordering in lists), and defines some new functions.

Definition

```
CREATE TYPE TEMPLATE  SQL_List ( :Element_Type TYPE )
                   UNDER SQL_Table ( :Element_Type )

       (

       -- default constructor and destructor signatures are to be inserted here
       -- when naming problem is resolved

       CAST ( :GEN_TYPE AS :Element_Type WITH Table_to_Element ) ; ,

       CAST ( :GEN_TYPE AS SQL_Table ( :Element_Type ) WITH List_to_Table ) ; ,

       CAST (:GEN_TYPE AS SQL_Set ( :Element_Type ) WITH Distinct ) ; ,

       CAST ( SQL_Empty_List AS :GENTYPE WITH Empty_List_to_List ) ; ,

       FUNCTION SQL_Insert ( :tabref REF ( :GEN_TYPE ) ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION SQL_Select ( :tabref REF (:GEN_TYPE ) ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) )
                RETURNS SQL_Table (Object) ; ,  !! assuming Object is a supertype of all
       ADTs

       FUNCTION SQL_Update ( :tabref REF ( :GEN_TYPE ) ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION SQL_Delete ( :tabref REF ( :GEN_TYPE ) ,
                             :stmt CHARACTER VARYING (max_stmt_length) ,
                             :descr_name CHARACTER VARYING (max_name_length) ) ; ,

       FUNCTION Position ( :list :GEN_TYPE, :elem :Element_Type )
                RETURNS INTEGER ; ,

       FUNCTION Element ( :list :GEN_TYPE, :pos INTEGER )
                RETURNS :Element_Type ; ,

       FUNCTION Head ( :list :GEN_TYPE ) RETURNS :Element_Type ;
                RETURN Element (:list, 1) ; ,
```

```
FUNCTION Tail ( :list :GEN_TYPE ) RETURNS :GEN_TYPE ; ,

FUNCTION Append ( :list :GEN_TYPE, :elem :Element_Type )
         RETURNS :GEN_TYPE ; ,

PRIVATE:

FUNCTION List_to_Table ( :set :GEN_TYPE )
         RETURNS SQL_Table ( :Element_Type ) ; ,

FUNCTION Empty_List_to_List ( :target :GEN_TYPE ) RETURNS :GEN_TYPE ;

) ;
```

Description

1) Let *T* be any data type. An instance of a generated type SQL_List(*T*) is a table, each of whose rows is of type *T*. The rows of an SQL_List have an ordinal position, beginning at 1 for the first element (row).

2) The value of *max_stmt_length* is an implementation-dependent maximum length for an <SQL procedure statement>. The value of *max_name_length* is the maximum length of a <descriptor name>.

   Note: I don't know how to fix this Rule, which refers to a <descriptor name>. Perhaps some part of it should be moved to Part *Z*, SQL/Bindings. Without knowing how to deal with this, I have simply copied the entire Description to Part *Z*.

3) Let *F* be an SQL_Insert, SQL_Select, SQL_Update, or SQL_Delete function. Let *T*, *S*, and *D* be the contents of the arguments passed to the :tabref, :stmt, and :descr_name parameters respectively of *F*. The behavior of *F*, *T*, *S*, and *D* is as in the description of the SQL_Table type template.

   Note: Note that this does not mean that the behavior is necessarily the same when *T* refers to an SQL_List. The description for an SQL_Table type template depends on the Format, Syntax Rules, Access Rules and General Rules of the statement whose character string representation is the value of the argumentS. The Format and Rules of such a statement differ, depending on whether the statement references an SQL_Table or an SQL_List.

4) The Position function returns the position of the first occurrence of the element specified by :elem in the SQL_List identified by :list, provided the element exists in that list; otherwise, the value 0 is returned.

5) The Element function returns the *i*-th element of the SQL_List identified by :list, where *i* is the value of :pos, provided that such an element exists; otherwise, an exception condition is raised: *SQL_List—invalid element position*.

6) The Head function returns the first element of :list.

7) The Tail function returns an SQL_List(*T*) that is a copy of :list with its first element, if any, removed.

8) The Append function returns an SQL_List(*T*) that is a copy of :list with :elem inserted as the last element.

9) The List_to_Table function returns an SQL_Table containing the same elements as the SQL_List.

10) The Empty_List_to_List function returns a list of data type SQL_List(*T*) that is empty, *i.e.*, contains no rows.

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no data type generators.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only—caused by ISO changes not yet considered by ANSI

G.2.6 SQL_Empty_Table type

Function

SQL_Empty_Table provides the ability to create an empty table, but not to alter its state.

Definition

```
CREATE TYPE SQL_Empty_Table WITHOUT OID
       (
         EQUALS DEFAULT;
         LESS_THAN NONE;

         CONSTRUCTOR FUNCTION SQL_Empty_Table ( ) RETURNS SQL_Empty_Table

       );
```

Description

1) An instance of SQL_Empty_Table is an always empty table, created either by explicit use of its constructor function, or by a <table value expression> containing <table type> TABLE and no <value expressions>s.

2) An instance of SQL_Empty_Table may also be cast to, assigned to, and compared with, an instance of any SQL_Table generated type (see Subclause G.2.3, "SQL_Table type template").

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type>s.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only—caused by ISO changes not yet considered by ANSI

G.2.7 SQL_Empty_Set type

Function

SQL_Empty_Set provides the ability to create an empty set, but not to alter its state.

Definition

```
CREATE TYPE SQL_Empty_Set WITHOUT OID
           UNDER SQL_Empty_Table
       (
         EQUALS DEFAULT;
         LESS_THAN NONE;

         CONSTRUCTOR FUNCTION SQL_Empty_Set ( ) RETURNS SQL_Empty_Set

       );
```

Description

1) An instance of SQL_Empty_Set is an always empty set, created either by explicit use of its constructor function, or by a <table value expression> containing <table type> SET and no <value expressions>s.

2) An instance of SQL_Empty_Set may also be cast to, assigned to, and compared with, an instance of any SQL_Set generated type (see Subclause G.2.4, "SQL_Set type template").

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type>s.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

ANSI Only—caused by ISO changes not yet considered by ANSI

G.2.8 SQL_Empty_List type

Function

SQL_Empty_List provides the ability to create an empty list, but not to alter its state.

Definition

```
CREATE TYPE SQL_Empty_List WITHOUT OID
           UNDER SQL_Empty_Table
       (
         EQUALS DEFAULT;
         LESS_THAN NONE;

         CONSTRUCTOR FUNCTION SQL_Empty_List ( ) RETURNS SQL_Empty_List

       );
```

Description

1) An instance of SQL_Empty_List is an always empty list, created either by explicit use of its constructor function, or by a <table value expression> containing <table type> LIST and no <value expressions>s.

2) An instance of SQL_Empty_List may also be cast to, assigned to, and compared with, an instance of any SQL_List generated type (see Subclause G.2.5, "SQL_List type template").

Leveling Rules

1) The following restrictions apply for Full SQL:

   a) Conforming Full SQL language shall contain no <abstract data type>s.

2) The following restrictions apply for Intermediate SQL in addition to any Full SQL restrictions:

   None.

3) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

Index

Index entries appearing in **boldface** indicate the page where the word, phrase, or BNF nonterminal was defined; index entries appearing in *italics* indicate a page where the BNF nonterminal was used in a Format; and index entries appearing in roman type indicate a page where the word, phrase, or BNF nonterminal was used in a heading, Function, Syntax Rule, Access Rule, General Rule, Leveling Rule, Table, or other descriptive text.

— B —

— C —

— D —

— E —

— F —

— G —

— H —

— I —

— J —

— K —

— L —

— M —

— N —

— O —

— P —

— Q —

— R —

— S —

— T —

— U —

— Y —

— Z —