dpANS (draft proposed American National Standard)
Database Language SQL
BSR X3.135-1992

January 1993

ANSI X3.135-1992, *Database Language SQL*—January 4, 1993

Digital Equipment Corporation
Maynard, Massachusetts

Contents

TABLES

Foreword

(This foreword is not a part of American National Standard X3.135-1992.)
This Standard (American National Standard X3.135-1992, *Database Language—SQL*), is a revision of American National Standard X3.135-1989, (*Database Language—SQL with Integrity Enhancement*), 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**).

Requests for interpretation, suggestions for improvement or addenda, or defect reports are welcome. They should be sent to the X3 Secretariat, Computer and Business Equipment Manufacturers Association, 1250 Eye Street, NW, Suite 200, Washington, DC 20005.

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

Mr. James D. Converse, Chair
Mr. Donald C. Loughry, Vice Chair
Ms. Joanne Flanagan, Secretary

Organization Represented	Name of Representative
Producer Group	
AMP Incorporated	Mr. Edward Kelly (P) Mr. Charles Brill (A)
AT&T/NCR Corporation	Mr. Thomas W. Kern (P) Mr. Thomas F. Frost (A)
Apple Computer, Inc.	Ms. Karen Higginbottom (P)
Compaq Computers	Mr. James Barnes (P) Mr. Keith Lucke (A)
Digital Equipment Corporation	Mr. Delbert Shoemaker (P) Mr. Kevin Lewis (A)
Hitachi America, Ltd.	Mr. John Neumann (P) Mr. Kei Yamashita (A)
Hewlett-Packard	Mr. Donald C. Loughry (P)
Bull HN Information Systems, Inc.	Mr. David M. Taylor (P)
IBM Corporation	Mr. Robert H. Follett (P) Ms. Mary Anne Lawler (A)
Unisys Corporation	Mr. John Hill (P) Mr. Stephen P. Oksala (A)
Sony Corporation of America	Mr. Michael Deese (P)

Storage Technology Corporation	Mr. Joseph S. Zajaczkowski (P) Mr. Samuel D. Cheatham (A)
Sun Microsystems, Inc.	Mr. Scott Jameson (P) Mr. Gary S. Robinson (A)
Xerox Corporation	Mr. Roy Pierce (P) Mr. Dwight McBain (A)
Wang Laboratories, Inc.	Mr. Steve Brody (P) Ms. Barbara Lurvey (A)
3M Company	Mr. Paul D. Jahnke (P)
Consumer Group	
Allen-Bradley Company	Mr. Ronald Reimer (P) Mr. Joe Lenner (A)
Boeing Company	Ms. Catherine Howells (P) Ms. Andrea Vanosdoll (A)
Digital Equipment Computer Users Society	Ms. Rochelle Lauer (P) Mr. Loren Buhle, Jr. (A)
Eastman Kodak Company	Mr. James Converse (P) Mr. Michael Nier (A)
General Services Administration	Mr. Douglas Arai (P) Mr. Larry L. Jackson (A)
Guide International, Inc.	Mr. Frank Kirshenbaum (P) Mr. Harold Kuneke (A)
Hughes Aircraft Company	Mr. Harold Zebrack (P)
Lawrence Berkeley Laboratory	Mr. Robert L. Fink (P) Mr. David F. Stevens (A)
National Communications Systems	Mr. Dennis Bodson (P) Mr. George W. White (A)
Northern Telecom, Inc.	Mr. Mel Woinsky (P) Mr. Subhash Patel (A)
Recognition Tech Users Association	Mr. Herbert P. Schantz (P) Mr. G. Edwin Hale (A)
Share, Inc.	Mr. Gary Ainsworth (P)
U.S. Department of Defense	Mr. William Rinehuls (P) Mr. Thomas Bozek (A)
U.S. Department of Energy	Mr. Alton Cox (P) Mr. Lawrence A. Wasson (A)
U.S. West Corporation	Mr. Gary Dempsey (P) Mr. Anislie Bates (A)
USE, Inc.	Mr. Peter Epstein (P)
Wintergreen Information Services	Mr. John Wheeler (P)
General Interest Group	
American Library Association	Mr. Paul Peters (P)
American Nuclear Society	Ms. Geraldine C. Main (P) Ms. Sally Hartzell (A)
Association of the Institute for Certification of Computer Professionals (AICCP)	Mr. Kenneth Zemrowski (P)

Electronic Data Systems Corporation	Mr. Charles M. Durrett (P) Mr. Jerrold S. Foley (A)
National Institute of Standards and Technology	Mr. Robert E. Rountree (P) Mr. Michael Hogan (A)
Omnicom, Inc.	Mr. Harold C. Folts (P) Ms. Kathleen Dally (A)
Open Systems Foundation (OSF)	Mr. Henry Lowe (P) Mr. Paul Rabin (A)

American National Standard X3.135-1992 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
Leonard J. Gallagher, International Representative
Michael M. Gorman, Secretary
Jim Melton, Editor

Paul Alley
Dean A. Anderson
John Barney
Rickie Brinegar
Amelia Carlson
Pedro Celis
Joe Celko
J. Douglas Conley
T. N. Doraiswamy
John Earle
Chris Farrar
Steve Felts
Lynn M. Francis
Wally Gazdzik
Tony Gordon
Keith Hare
Rebecca Harris
Merrill Holt
Jeff Horowitz
Bob Huntsman
Ken Jacobs
Phil Jones
Carol D. Joyce
Eugene L. Kaplan
Setrag Khoshafian
Aviel Klausner
Tom Lavoie
Dennis Layton
Geoffrey Lyman
John K. Lyon
David Markham
Jeff Mischkinsky
Alex Mittelman

Michael Pantaleo
Robin Pasley
Gail Penrod
Paul Perkovic
Laila G. Robinson
John Sadd
Larry Settle
Phil Shaw
Michael J. Simko
Harvey Smith
C. Stephen Smyth
Rick Stellwagen
Jagen Nath Sud
Emy Tseng
Boris Tsukerman
Richard Tucker
Barry D. Vickers
Philip Welsh
Laura Yedwab
Robert Zeek

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

Razmik Abnous
Roberta Allen
Jerome L. Althoff
Bradford Anderson
Robert D. Atlas
Jerry Baker
Mike Bashir
Jonathan Bauer
Nick Baxter
Rob Bell
Alan Bier
Jarvis A. Boykin
Bill R. Brykczynski
Arvola Chan
Sanjoy Chatterjee
John Dove
Andrew J. Eisenberg
Marco Emrich
Derek Frankforth
Jeffrey Fried
Fred Friedman
Barry Fritchman
Michael D. Gagle
Jack Hahn
Carl L. Hall
Alan R. Hirsch
Jeff Jones
Steve Jones
Tom Julien
Hugh Jurkiewicz

Sue Karlin
Michael W. Kelley
Paul Kent
Steve Klein
Bernard Kocis
Rona J. Kopp
William J. Koster
May Kovalick
Dominique Laborde
John Lax
Dennsi Leatherwood
Jeffrey Lichtman
Don Lind
William E. Linn
Mark Lipp
James Lofstrand
Louise M. Madrid
Jerry Mara
Anthony R. Marriott
James Moore
Kenneth G. Moore
Mark E. Moore
Alan Nall
Phil Neches
Kenneth Ng
Chuck Olson
Ken Paris
Richard Pedigo
William Phillips
William E. Raab
James Richardson
Robert Rittler
Tom Rogers
Dave Ross
Roger Schuelke
Edwin Seputis
Bruce Shenker
Hunter H. Shu
Val Skalabrin
William Stoeller
Joan M. Sullivan
Tibor Vais
Elaine Volkman
John R. Walsh
Chip Ziering

This American Standard contains seven 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 ANSI X3.135-1989; and

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

Introduction

This American Standard was approved in 1992.

This American Standard was developed from ANSI X3.135-1989, Information Systems, Database Language SQL with Integrity Enhancements, and replaces that American Standard. It adds significant new features and capabilities to the specifications. It is generally compatible with ANSI X3.135-1989 in the sense that, with very few exceptions, SQL language that conforms to ANSI X3.135-1989 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 ANSI X3.135-1989. The known incompatibilities between ANSI X3.135-1989 and this American Standard are stated in informative Annex E, "Incompatibilities with ANSI X3.135-1989".

Technical changes between ANSI X3.135-1989 and this American Standard include both improvements or enhancements to existing features and the definition of new features. Significant improvements in existing features include:

— A better definition of direct invocation of SQL language;

— Improved diagnostic capabilities, especially a new status parameter (SQLSTATE), a diagnostics area, and supporting statements.

Significant new features are:

1) Support for additional data types (DATE, TIME, TIMESTAMP, INTERVAL, BIT string, variable-length character and bit strings, and NATIONAL CHARACTER strings),

2) Support for character sets beyond that required to express SQL language itself and support for additional collations,

3) Support for additional scalar operations, such as string operations for concatenate and substring, date and time operations, and a form for conditional expressions,

4) Increased generality and orthogonality in the use of scalar-valued and table-valued query expressions,

5) Additional set operators (for example, union join, natural join, set difference, and set intersection),

6) Capability for domain definitions in the schema,

7) Support for Schema Manipulation capabilities (especially DROP and ALTER statements),

8) Support for bindings (modules and embedded syntax) in the MUMPS language,

9) Additional privilege capabilities,

10) Additional referential integrity facilities, including referential actions, subqueries in CHECK constraints, separate assertions, and user-controlled deferral of constraints,

11) Definition of an Information Schema,

12) Support for dynamic execution of SQL language,

13) Support for certain facilities required for Remote Database Access (especially connection management statements and qualified schema names),

14) Support for temporary tables,

15) Support for transaction consistency levels,

16) Support for data type conversions (CAST expressions among data types),

17) Support for scrolled cursors, and

18) A requirement for a flagging capability to aid in portability of application programs.

The organization of this American Standard is as follows:

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

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

3) Clause 3, "Definitions, notations, and conventions", defines the notations and conventions used in this American 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", 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, "Dynamic SQL", defines the facilities for executing SQL-statements dynamically.

18) Clause 18, "Diagnostics management", defines the diagnostics management facilities.

19) Clause 19, "Embedded SQL", defines syntax for embedding SQL in certain standard programming languages.

20) Clause 20, "Direct invocation of SQL", defines the direct invocation of SQL language.

21) Clause 21, "Information Schema and Definition Schema", defines viewed tables that contain schema information.

22) Clause 22, "Status codes", defines values that identify the status of the execution of SQL-statements and the mechanisms by which those values are returned.

23) Clause 23, "Conformance", defines the criteria for conformance to this American standard.

24) Annex A, "Leveling the SQL Language", is an informative Annex. It lists the leveling rules defining the Entry SQL and Intermediate SQL subset levels of the SQL language.

25) Annex B, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of the American 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.

26) Annex C, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of the American Standard states explicitly that the meaning or effect on the database is implementation-dependent.

27) Annex D, "Deprecated features", is an informative Annex. It lists features that the responsible Technical Committee intends will not appear in a future revised version of this American Standard.

28) Annex E, "Incompatibilities with ANSI X3.135-1989", is an informative Annex. It lists the incompatibilities between this version of this American Standard and ANSI X3.135-1989.

29) Annex F, "Maintenance and interpretation of SQL", is an informative Annex. It identifies SQL interpretations and corrections that have been processed by ANSI Accredited Committee X3 since adoption of ANSI X3.135-1989.

In the text of this American Standard, Clauses begin a new odd-numbered page, and in Clause 5, "Lexical elements", through Clause 22, "Status codes", Subclauses begin a new page. Any resulting blank space is not significant.

Information Technology — Database Language SQL

1 Scope

This American 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 American Standard is described by the Reference Model of Data Management (ISO/IEC DIS 10032:1991).

This American 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 and embedding them into a standard programming language.

It also specifies an Information Schema that describes the structure and the integrity constraints of SQL-data.

This American Standard

— provides a vehicle for portability of data definitions and compilation units between SQL-implementations,

— provides a vehicle for interconnection of SQL-implementations,

— specifies syntax for embedding SQL-statements in a compilation unit that otherwise conforms to the standard for a particular programming language. It defines how an equivalent compilation unit may be derived that conforms to the particular programming language standard. In that equivalent compilation unit, each embedded SQL-statement has been replaced by statements that invoke a database language procedure that contains the SQL-statement, and

— specifies syntax for direct invocation of SQL-statements.

This American 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, including those containing embedded SQL.

Implementations of this American 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 American Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this American 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 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 American 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/IEC 8824:1990, *Information technology—Open Systems Interconnection—Specification of Abstract Syntax Notation One (ASN.1)*.

— ISO/IEC 9579-2:[1], *Information technology – Open Systems Interconnection — Remote Database Access, Part 2: SQL specialization*.

— ISO/IEC 10646:[1], *Information technology—Multiple-octet coded character set*.

[1] To be published

4 Database Language SQL

3 Definitions, notations, and conventions

3.1 Definitions

For the purposes of this American Standard, the following definitions apply.

3.1.1 Definitions taken from ISO/IEC 10646

This American 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 American 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 American Standard

This American Standard defines the following terms:

a) **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.

b) **cardinality** (of a collection): The number of objects in that collection. Those objects need not necessarily have distinct values.

c) **character repertoire**; **repertoire**: A set of characters used for a specific purpose or application. Each character repertoire has an implied default collating sequence.

d) **coercibility**: An attribute of character string data items that governs how a collating sequence for the items is determined.

e) **collation**; **collating sequence**: A method of ordering two comparable character strings. Every character set has a default collation.

f) **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.

g) **descriptor**: A coded description of an SQL object. It includes all of the information about the object that a conforming SQL-implementation requires.

h) **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.

i) **duplicate**: Two or more values or rows are said to be duplicates (of each other) if and only if they are not distinct.

j) **dyadic operator**: An operator having two operands: a left operand and a right operand. An example of a dyadic arithmetic operator in this American Standard is "–", specifying the subtraction of the right operand from the left operand.

k) **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.

l) **form-of-use conversion**: A method of converting character strings from one form-of-use to another form-of-use.

m) **implementation-defined**: Possibly differing between SQL-implementations, but specified by the implementor for each particular SQL-implementation.

n) **implementation-dependent**: Possibly differing between SQL-implementations, but not specified by this American Standard and not required to be specified by the implementor for any particular SQL-implementations.

o) **monadic operator**: An operator having one operand. An example of a monadic arithmetic operator in this American Standard is "–", specifying the negation of the operand.

p) **multiset**: An unordered collection of objects that are not necessarily distinct. The collection may be empty.

q) ***n*-adic operator**: An operator having a variable number of operands (informally: *n* operands). An example of an *n*-adic operator in this American Standard is COALESCE.

r) **null value (null)**: A special value, or mark, that is used to indicate the absence of any data value.

s) **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 4.28, "SQL-transactions") or an SQL-session (see Subclause 4.30, "SQL-sessions") are not regarded as deliberate.

t) **redundant duplicates**: All except one of any multiset of duplicate values or rows.

u) **repertoire**: See **character repertoire**.

v) **sequence**: An ordered collection of objects that are not necessarily distinct.

w) **set**: An unordered collection of distinct objects. The collection may be empty.

x) **SQL-implementation**: A database management system that conforms to this American Standard.

y) **translation**: A method of translating characters in one character repertoire into characters of the same or a different character repertoire.

3.2 Notation

The syntactic notation used in this American Standard is an extended version of BNF ("Backus Naur Form" or "Backus Normal 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 American 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 shall 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 ordinary 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 American 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 American 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 shall 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 numerical 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 shall 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 American 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 12.3, "<procedure>", and Subclause 20.1, "<direct SQL statement>", may be terminated, diagnostic information is to be made available, and 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 American 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.

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 <B>, then <A> generally contains <B>. If <A> generally contains <B> and <B> generally contains <C>, then <A> generally contains <C>.

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* (if this situation occurs during dynamic execution of an SQL-statement, then the exception that is raised is *syntax error or access rule violation in dynamic SQL statement*; if the situation occurs during direct invocation of an SQL-statement, then the exception that is raised is *syntax error or access rule violation in direct SQL statement*).

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* (if this situation occurs during dynamic execution of an SQL-statement, then the exception that is raised is *syntax error or access rule violation in dynamic SQL statement*; if the situation occurs during direct invocation of an SQL-statement, then the exception that is raised is *syntax error or access rule violation in direct SQL statement*).

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 American 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.

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 <procedure> that has been executed, or if *S1* is the value of an <SQL statement variable> referenced by an <execute immediate statement> contained in a <procedure> that has been executed, or if *S1* was the value of the <SQL statement variable> that was associated with an <SQL statement name> by a <prepare statement> and that same <SQL statement name> is referenced by an <execute statement> contained in a <procedure> that has been executed.

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* zero or more table descriptors (equivalently, an assertion *is dependent on* or *depends on* zero 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>, *etc.* 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 American 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>
```

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

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.

A value is primitive in that it has no logical subdivision within this American Standard. A value is a null value or a non-null value.

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, BIT, BIT VARYING, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION, DATE, TIME, TIMESTAMP, and INTERVAL.

Subclause 6.1, "<data type>", describes the semantic properties of each data type.

For reference purposes, the data types CHARACTER and CHARACTER VARYING are collectively referred to as character string types. The data types BIT and BIT VARYING are collectively referred to as bit string types. Character string types and bit 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 type 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.

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 12.3, "<procedure>", and Subclause 19.1, "<embedded SQL host program>". 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 or CHARACTER VARYING; NATIONAL CHARACTER and NATIONAL CHARACTER VARYING are represented as CHARACTER and CHARACTER VARYING, 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 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 mutually comparable, subject to the restrictions specified in Table 3, "Collating sequence usage for comparisons".

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

— the name of the default collation of the character set.

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 set on which the collation operates,

— whether the collation has the NO PAD or the PAD SPACE attribute, and

— an indication of how the collation is performed.

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.

<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 American 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 integer, 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.

<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.

4.2.3 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 10.4, "<character set specification>", and Subclause 11.28, "<character set definition>".

— When columns are involved (*e.g.*, comparing two columns, or comparing a column to a literal), by default the default collating sequence of the columns involved is used so long as the columns have the same default collating sequence.

— When columns are involved having different default collating sequences, explicit specification of the collating sequence in the expression is required via the <collate clause> when the expression participates in a comparison.

— 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 *Implicit* attribute, 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 *Coercible* attribute, with the default collation for its character repertoire. A <character value expression> simply containing a <collate clause> has the *Explicit* attribute, 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*

Table 2—Collating coercibility rules for dyadic operators (Cont.)

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**
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 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).

4.3.1 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>".

Assignment of a bit string to a bit string variable is performed from the most significant bit to the least significant bit in the source string to the most significant bit in the target string, one bit at a time.

4.3.2 Operations involving bit strings

4.3.2.1 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.

4.3.2.2 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.

4.4 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.

4.4.1 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 $10^{exponent}$.

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 numerical value *N* for an <exact numeric type> *T* is a value *V* representable in *T* such that *N* is not closer to zero than the numerical value of *V* and such that the absolute value of the difference between *N* and the numerical value of *V* is less than the absolute value of the difference between two successive numerical values representable in *T*.

An approximation obtained by rounding of a numerical value *N* for an <exact numeric type> *T* is a value *V* representable in *T* such that the absolute value of the difference between *N* and the numerical value of *V* is not greater than half the absolute value of the difference between two successive numerical values representable in *T*. If there are more than one such values *V*, then it is implementation-defined which one is taken.

All numerical values between the smallest and the largest value, inclusive, representable in a given exact numeric type have an approximation obtained by rounding or truncation for that type; it is implementation-defined which other numerical values have such approximations.

An approximation obtained by truncation or rounding of a numerical value *N* for an <approximate numeric type> *T* is a value *V* representable in *T* such that there is no numerical value representable in *T* and distinct from that of *V* that lies between the numerical value of *V* and *N*, inclusive.

If there are more than one such values *V* then it is implementation-defined which one is taken. It is implementation-defined which numerical 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 6.12, "<numeric value expression>".

4.4.2 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 4.3.2, "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 4.3.2, "Operations involving bit strings") operates on a string argument and returns an integer;

— <extract expression> (see Subclause 4.5.3, "Operations involving datetimes and intervals") operates on a datetime or interval argument and returns an integer.

4.5 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.

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.

4.5.1 Datetimes

Table 4, "Fields in datetime items", specifies the fields that can make up a date time 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 American 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.

4.5.2 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 subsequence 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 or the least significant end of one or both day-time intervals. 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.

4.5.3 Operations involving datetimes and intervals

Table 8, "Valid operators involving datetimes and intervals", specifies 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.

4.6 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 6.12, "<numeric value expression>".

Values corresponding to the data types CHARACTER and CHARACTER VARYING 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 shall 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 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 to be 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 less than a 1-bit.

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.

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.

4.7 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 value of <default option>, if any, of the domain; and

— the domain constraint descriptors of the domain constraints, if any, of the domain.

4.8 Columns

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.

Every column has a <column name>.

Every column has a nullability characteristic of *known not nullable* or *possibly nullable*, defined as follows:

A column has a nullability characteristic that indicates whether any attempt to store a null value into that column will inevitably raise an exception, or whether any attempt to retrieve a value from that column can ever result in a null value. 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 expression> 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.

— *CN* is contained in a non-deferrable <unique constraint definition> whose <unique specification> specifies PRIMARY KEY.

Otherwise, a column *C* is *possibly nullable*.

A column 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 the column;

— the <collation name> from the <collate clause>, if any, of the column;

— the value of <default option>, if any, of the column;

— the nullability characteristic of the column; and

— the ordinal position of the column within the table that contains the column.

4.9 Tables

A table is a multiset of rows. A row is a nonempty sequence of values. Every row of the same table has the same cardinality and contains a value of every column of that table. The *i*-th value in every row of a table is a value of the *i*-th column of that table. The row is the smallest unit of data that can be inserted into a table and deleted from a table.

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, a viewed 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 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 7.9, "<query specification>", Subclause 7.10, "<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 updatable. Derived tables are either updatable or read-only. The operations of insert, update, and delete are permitted for updatable tables, subject to constraining Access Rules. The operations of insert, update, and delete are not allowed for read-only tables.

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 a named table defined by a <temporary table declaration> that is effectively materialized the first time any <procedure> in the <module> that contains the <temporary table declaration> is executed. A declared local temporary table is accessible only by <procedure>s in the <module> that contains the <temporary table declaration>. The effective <schema name> of the <qualified name> of the declared local temporary table may be thought of as the implementation-dependent SQL-session identifier associated with the SQL-session and a unique implementation-dependent name associated with the <module> that contains the <temporary table declaration>. All references to a declared local temporary table are prefixed by "MODULE.".

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 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:

— 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; and

— the descriptor of each table constraint specified for the table.

A derived table descriptor describes a derived table. In addition to the components of every table descriptor, a derived table descriptor includes:

— if the table is named, then the name of the table;

— the <query expression> that defines how the table is to be derived; and

— an indication of whether the derived table is updatable or read-only (this is derived from the <query expression>);

A view descriptor describes a view. In addition to the components of a derived table descriptor, a view descriptor includes:

— an indication of whether the view has the CHECK OPTION; if so, whether it is to be applied as CASCADED or LOCAL.

4.10 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.

4.10.1 Checking of constraints

Every constraint is either *deferrable* or *non-deferrable*. Within a transaction, every constraint has a constraint mode; if a constraint is *non-deferrable*, then its constraint mode is always *immediate*, otherwise it is either or *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>.

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 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 transaction is terminated by an implicit <rollback statement>.

4.10.2 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>, 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 the 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.

4.10.3 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.

4.10.4 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.

4.11 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; and

— the descriptor of every component of the SQL-schema.

In this American 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, an assertion descriptor, a privilege descriptor, a character set descriptor, a collation descriptor, or a translation 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 <qualified name>. The default schema for <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s is initially implementation-defined but may be changed by the use of <set schema statement>s.

4.12 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. The default catalog for <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s is initially implementation-defined but may be changed by the use of <set catalog statement>s.

4.13 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 definition 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 definition> and no <joined table> can "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.

4.14 SQL-data

SQL-data is any data described by schemas that is under the control of an SQL-implementation in an SQL-environment.

4.15 SQL-environment

An *SQL-environment* comprises the following:

— an SQL-implementation capable of processing some Level (Entry SQL, Intermediate SQL, or Full SQL) of this American Standard and at least one binding style; see Clause 23, "Conformance" for further information about binding styles;

— 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.

4.16 Modules

A <module> is an object specified in the module language. A <module> is either a persistent <module> or an SQL-session <module>. The mechanisms by which <module>s are created or destroyed are implementation-defined. A <module> consists of an optional <module name>, a <language clause>, a <module authorization clause> with either or both of a <module authorization identifier> and a <schema name>, 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 cursors specified by <declare cursor>s, and one or more <procedure>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.

4.17 Procedures

A <procedure> consists of a <procedure name>, a sequence of <parameter declaration>s, and a single <SQL procedure statement>.

A <procedure> in a <module> is invoked by a compilation unit associated with the <module> by means of a host language "call" statement that specifies the <procedure name> of the <procedure> and supplies a sequence of parameter values corresponding in number and in <data type> to the <parameter declaration>s of the <procedure>. A call of a <procedure> causes the <SQL procedure statement> that it contains to be executed.

4.18 Parameters

A parameter is declared in a <procedure> by a <parameter declaration>. The <parameter declaration> specifies the <data type> of its value. A parameter either assumes or supplies the value of the corresponding argument in the call of that <procedure>. These <data type>s map to host language types and are not nullable except through the use of additional indicator variables.

4.18.1 Status parameters

The SQLSTATE and SQLCODE parameters are status parameters. They are set to status codes that indicate either that a call of the <procedure> completed successfully or that an exception condition was raised during execution of the <procedure>.

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 American Standard. See Annex D, "Deprecated features".

A <procedure> shall 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 22, "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*.

4.18.2 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.

4.18.3 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. 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 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, then the indicator parameter or variable is set to the length of the source data item (in characters or bits, as appropriate) to indicate that truncation occurred and to indicate the original length in characters or bits, as appropriate, of the source.

4.19 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 statement that is not an <SQL diagnostics statement>, the diagnostics area is emptied. An implementation shall 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.

<procedure>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.

4.20 Standard programming languages

This American Standard specifies the actions of <procedure>s in <module>s when those <procedure>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". This American Standard also specifies a mechanism whereby SQL language may be embedded in programs that otherwise conform to any of the same specified programming language standards.

Note: In this American Standard, for the purposes of interfacing with programming languages, the data types DATE, TIME, TIMESTAMP, and INTERVAL shall 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 shall 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 this American 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 19.1, "<embedded SQL host program>". For standard programming languages Ada and PL/I, bit string variables are directly supported.

4.21 Cursors

A cursor is specified by a <declare cursor>, <dynamic declare cursor>, or <allocate cursor statement>.

For every <declare cursor> or <dynamic declare cursor> in a <module>, a cursor is effectively created when an SQL-transaction (see Subclause 4.28, "SQL-transactions") referencing the <module> is initiated, and destroyed when that SQL-transaction is terminated. A cursor is also effectively created when an <allocate cursor statement> is executed within a SQL-transaction and destroyed when that SQL-transaction is terminated. In addition, an extended dynamic cursor is destroyed when a <deallocate prepared statement> is executed that deallocates the prepared statement on which the extended dynamic cursor is based.

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> or <dynamic open statement> and returned to the closed state by a <close statement> or <dynamic close statement>, a <commit statement>, or a <rollback 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 positions 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 *read-only* or *updatable*. If the table identified by a cursor is not updatable or if INSENSITIVE is specified for the cursor, then the cursor is read-only; otherwise, the cursor is updatable. The operations of update and delete are not allowed for read-only cursors.

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 <fetch statement> or <dynamic 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> or <dynamic update statement: positioned> updates the current row of the cursor. A <delete statement: positioned> or <dynamic delete statement: positioned> deletes the current row of the cursor.

If an error occurs during the execution of an SQL-statement that identifies an open cursor, then, except where otherwise explicitly defined, the effect, if any, on the position or state of that cursor is implementation-dependent.

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.

4.22 SQL-statements

4.22.1 Classes of SQL-statements

An SQL-statement is a string of characters that conforms to the format and syntax rules specified in this international standard. Most SQL-statements can be prepared for execution and executed in one of a number of ways. These are:

— in a <module>, in which case it is prepared when the <module> is created (see Subclause 4.16, "Modules") and executed when the containing procedure is called.

— in an embedded SQL host program, in which case it is prepared when the embedded SQL host program is preprocessed (see Subclause 4.23, "Embedded syntax").

— being prepared and executed by the use of SQL-dynamic statements (which are themselves executed in one of the foregoing two ways—see Subclause 4.24, "SQL dynamic statements").

— direct invocation, in which case it is effectively prepared immediately prior to execution (see Subclause 4.25, "Direct invocation of SQL").

There are at least five ways of classifying SQL-statements:

— According to their effect on SQL objects, whether persistent objects, *i.e.*, SQL-data and schemas, or transient objects, such as SQL-sessions and other SQL-statements.

— According to whether or not they start a transaction, or can, or must, be executed when no transaction is active.

— According to whether or not they may be embedded.

— According to whether they may be dynamically prepared and executed.

— According to whether or not they may be directly executed.

This American 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.

4.22.2 SQL-statements classified by function

The following are the main classes of SQL-statements:

— SQL-schema statements; these may have a persistent effect on 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-connection statements

— SQL-session statements

— SQL-dynamic statements

— SQL-diagnostics statements

— SQL embedded exception declaration

The following are the SQL-schema statements:

— <schema definition>

— <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>

The following are the SQL-data statements:

— <temporary table declaration>

— <declare cursor>

— <dynamic declare cursor>

— <allocate cursor statement>

— <dynamic select statement>

— <open statement>

— <dynamic open statement>

— <close statement>

— <dynamic close statement>

— <fetch statement>

— <dynamic fetch statement>

— <select statement: single row>

— <direct select statement: multiple rows>

— <dynamic single row select statement>

— All SQL-data change statements

The following are the SQL-data change statements:

— <insert statement>

— <delete statement: searched>

— <delete statement: positioned>

— <dynamic delete statement: positioned>

— <preparable dynamic delete statement: positioned>

— <update statement: searched>

— <update statement: positioned>

— <dynamic update statement: positioned>

— <preparable dynamic update statement: positioned>

The following are the SQL-transaction statements:

— <set transaction statement>

— <set constraints mode statement>

— <commit statement>

— <rollback statement>

The following are the SQL-connection statements:

— <connect statement>

— <set connection statement>

— <disconnect statement>

The following are the SQL-session statements:

— <set catalog statement>

— <set schema statement>

— <set names statement>

— <set session authorization identifier statement>

— <set local time zone statement>

The following are the SQL-dynamic statements:

— <execute immediate statement>

— <allocate descriptor statement>

— <deallocate descriptor statement>

— <get descriptor statement>

— <set descriptor statement>

— <prepare statement>

— <deallocate prepared statement>

— <describe input statement>

— <describe output statement>

— <execute statement>

The following is the SQL-diagnostics statement:

— <get diagnostics statement>

The following is the SQL embedded exception declaration:

— <embedded exception declaration>

4.22.3 Embeddable SQL-statements

The following SQL-statements are embeddable in an embedded SQL host program, and may be the <SQL procedure statement> in a <procedure> in a <module>:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-connection statements

— All SQL-session statements

— All SQL-dynamic statements

— All SQL-diagnostics statements

— The following SQL-data statements:

- <allocate cursor statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>
- <select statement: single row>
- <insert statement>
- <delete statement: searched>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <update statement: searched>
- <update statement: positioned>

- <dynamic update statement: positioned>

The following SQL-statements are embeddable in an embedded SQL host program, and may occur in a <module>, though not in a <procedure>:

— <temporary table declaration>

— <declare cursor>

— <dynamic declare cursor>

The following SQL-statements are embeddable in an embedded SQL host program, but may not occur in a <module>:

— SQL embedded exception declarations

Consequently, the following SQL-data statements are not embeddable in an embedded SQL host program, nor may they occur in a <module>, nor be the <SQL procedure statement> in a <procedure> in a <module>:

— <dynamic select statement>

— <dynamic single row select statement>

— <direct select statement: multiple rows>

— <preparable dynamic delete statement: positioned>

— <preparable dynamic update statement: positioned>

4.22.4 Preparable and immediately executable SQL-statements

The following SQL-statements are preparable:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-session statements

— The following SQL-data statements:

- <delete statement: searched>
- <dynamic select statement>
- <dynamic single row select statement>
- <insert statement>
- <update statement: searched>
- <preparable dynamic delete statement: positioned>
- <preparable dynamic update statement: positioned>

- <preparable implementation-defined statement>

Consequently, the following SQL-statements are not preparable:

— All SQL-connection statements

— All SQL-dynamic statements

— All SQL-diagnostics statements

— SQL embedded exception declarations

— The following SQL-data statements:

  - <allocate cursor statement>
  - <open statement>
  - <dynamic open statement>
  - <close statement>
  - <dynamic close statement>
  - <fetch statement>
  - <dynamic fetch statement>
  - <select statement: single row>
  - <delete statement: positioned>
  - <dynamic delete statement: positioned>
  - <update statement: positioned>
  - <dynamic update statement: positioned>
  - <direct select statement: multiple rows>
  - <temporary table declaration>
  - <declare cursor>
  - <dynamic declare cursor>

Any preparable SQL-statement can be executed immediately, with the exception of:

— <dynamic select statement>

— <dynamic single row select statement>

4.22.5 Directly executable SQL-statements

The following SQL-statements may be executed directly:

— All SQL-schema statements

— All SQL-transaction statements

— All SQL-connection statements

— All SQL-session statements

— The following SQL-data statements:

- <temporary table declaration>
- <direct select statement: multiple rows>
- <insert statement>
- <delete statement: searched>
- <update statement: searched>

Consequently, the following SQL-statements may not be executed directly:

— All SQL-dynamic statements

— All SQL-diagnostics statements

— SQL embedded exception declarations

— The following SQL-data statements:

- <declare cursor>
- <dynamic declare cursor>
- <allocate cursor statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>
- <select statement: single row>
- <dynamic select statement>
- <dynamic single row select statement>

- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <preparable dynamic delete statement: positioned>
- <update statement: positioned>
- <dynamic update statement: positioned>
- <preparable dynamic update statement: positioned>

4.22.6 SQL-statements and transaction states

Whether an <execute immediate statement> starts a transaction depends on what SQL-statement is the value of <SQL statement variable>. Whether an <execute statement> starts a transaction depends on what SQL-statement was the value of <SQL statement variable> when the prepared statement identified by <SQL statement name> was prepared.

The following SQL-statements are transaction initiating SQL-statements, *i.e.*, if there is no current transaction, and a statement of this class is executed, a transaction is initiated:

— All SQL-schema statements

— The following SQL-data statements:

- <allocate cursor statement>
- <dynamic select statement>
- <open statement>
- <dynamic open statement>
- <close statement>
- <dynamic close statement>
- <fetch statement>
- <dynamic fetch statement>
- <select statement: single row>
- <direct select statement: multiple rows>
- <dynamic single row select statement>
- <insert statement>
- <delete statement: searched>
- <delete statement: positioned>
- <dynamic delete statement: positioned>
- <preparable dynamic delete statement: positioned>

- <update statement: searched>
- <update statement: positioned>
- <dynamic update statement: positioned>
- <preparable dynamic update statement: positioned>

— The following SQL-dynamic statements:

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

The following SQL-statements are not transaction initiating SQL-statements, *i.e.*, if there is no current transaction, and a statement of this class is executed, no transaction is initiated.

— All SQL-transaction statements

— All SQL-connection statements

— All SQL-session statements

— All SQL-diagnostics statements

— SQL embedded exception declarations

— The following SQL-data statements:

- <temporary table declaration>
- <declare cursor>
- <dynamic declare cursor>
- <dynamic select statement>

4.23 Embedded syntax

An <embedded SQL host program> (<embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL COBOL program>, <embedded SQL Fortran program>, <embedded SQL MUMPS program>, <embedded SQL Pascal program>, or <embedded SQL PL/I program>) is a compilation unit that consists of programming language text and SQL text. The programming language text shall conform to the requirements of a specific standard programming language. The SQL text

shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s, as defined in this American Standard. This allows database applications to be expressed in a hybrid form in which SQL-statements are embedded directly in a compilation unit. Such a hybrid compilation unit is defined to be equivalent to a standard compilation unit in which the SQL-statements have been replaced by standard procedure or subroutine calls of SQL <procedure>s in a separate SQL <module>, and in which each <embedded SQL begin declare> and each <embedded SQL end declare> has been removed and the declarations contained therein have been suitably transformed into standard host-language syntax.

An implementation may reserve a portion of the name space in the <embedded SQL host program> for the names of procedures or subroutines that are generated to replace SQL-statements and for program variables and branch labels that may be generated as required to support the calling of these procedures or subroutines; whether this reservation is made is implementation-defined. They may similarly reserve name space for the <module name> and <procedure name>s of the generated <module> that may be associated with the resulting standard compilation unit. The portion of the name space to be so reserved, if any, is implementation-defined.

4.24 SQL dynamic statements

In many cases, the SQL-statement to be executed can be coded into a <module> or into a compilation unit using the embedded syntax. In other cases, the SQL-statement is not known when the program is written and will be generated during program execution. An <execute immediate statement> can be used for a one-time preparation and execution of an SQL-statement. A <prepare statement> is used to prepare the generated SQL-statement for subsequent execution. A <deallocate prepared statement> is used to deallocate SQL-statements that have been prepared with a <prepare statement>. A description of the input parameters for a prepared statement can be obtained by execution of a <describe input statement>. A description of the resultant columns of a <dynamic select statement> or <dynamic single row select statement> can be obtained by execution of a <describe output statement>. For a statement other than a <dynamic select statement>, an <execute statement> is used to associate parameters with the prepared statement and execute it as though it had been coded when the program was written. For a <dynamic select statement>, the prepared <cursor specification> is associated with a cursor via a <dynamic declare cursor> or <allocate cursor statement>. The cursor can be opened and parameters can be associated with the cursor with a <dynamic open statement>. A <dynamic fetch statement> positions an open cursor on a specified row and retrieves the values of the columns of that row. A <dynamic close statement> closes a cursor that was opened with a <dynamic open statement>. A <dynamic delete statement: positioned> is used to delete rows through a dynamic cursor. A <dynamic update statement: positioned> is used to update rows through a dynamic cursor. A <preparable dynamic delete statement: positioned> is used to delete rows through a dynamic cursor when the precise format of the statement isn't known until runtime. A <preparable dynamic update statement: positioned> is used to update rows through a dynamic cursor when the precise format of the statement isn't known until runtime.

The interface for input parameters for a prepared statement and for the resulting values from a <dynamic fetch statement> or the execution of a prepared <dynamic single row select statement> can be either a list of parameters or embedded variables or an SQL descriptor area. An SQL descriptor area consists of zero or more item descriptor areas, together with a COUNT of the number of those item descriptor areas. Each item descriptor area consists of the fields specified in Table 17, "Data types of <key word>s used in SQL item descriptor areas", in Subclause 17.1, "Description of SQL item descriptor areas". The SQL descriptor area is allocated and maintained by the system with the following statements: <allocate descriptor statement>, <deallocate descriptor statement>, <set descriptor statement>, and <get descriptor statement>.

4.24 SQL dynamic statements

An SQL descriptor area is identified by a <descriptor name>, which is a <simple value specification> whose value is an <identifier>. Two <descriptor name>s identify the same SQL descriptor area if their values, with leading and trailing <space>s removed, are equivalent according to the rules for <identifier> comparisons in Subclause 5.2, "<token> and <separator>".

Dynamic statements can be identified by <statement name>s or by <extended statement name>s. Similarly, dynamic cursors can be identified by <cursor name>s and by <extended cursor name>s. The non-extended names are <identifier>s. The extended names are <target specification>s whose values are <identifier>s used to identify the statement or cursor. Two extended names are equivalent 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>".

An SQL descriptor area name may be defined as global or local. Similarly, an extended statement name or extended cursor name may be global or local. The scope of a global name is the SQL-session. The scope of a local name is the <module> in which it appears. A reference to an entity in which one specifies a global scope is valid only if the entity was defined as global and if the reference is from the same SQL-session in which it was defined. A reference to an entity in which one specifies a local scope is valid only if the entity was defined as local and if the reference is from the same <module> in which it was defined. (The scope of non-extended statement names and non-extended cursor names is always local.)

Within an SQL-session, all global prepared statements (prepared statements with global statement names) belong to the SQL-session <module>. Within an SQL-session, each local prepared statement (prepared statements with local statement names) belongs to the <module> that contains the <prepare statement> or <execute immediate statement> with which it is prepared.

Note: The SQL-session <module> is defined in Subclause 4.30, "SQL-sessions".

Dynamic execution of SQL-statements can generally be accomplished in two different ways. Statements can be *prepared* for execution and then later executed one or more times; when the statement is no longer needed for execution, it can be *released* by the use of a <deallocate prepared statement>. Alternatively, a statement that is needed only once can be executed without the preparation step—it can be *executed immediately* (not all SQL-statements can be executed immediately).

Many SQL-statements can be written to use "parameters" (which are manifested in static execution of SQL-statements as <parameters> in <SQL statement>s contained in <procedure>s in <module>s or as <embedded variable name>s in <SQL statement>s contained in <embedded SQL host program>s). In SQL-statements that are executed dynamically, the parameters are called dynamic parameters (<dynamic parameter specification>s) and are represented in SQL language by a <question mark> (?).

In many situations, an application that generates an SQL-statement for dynamic execution knows in detail the required characteristics (*e.g.*, <data type>, <length>, <precision>, <scale>, *etc.*) of each of the dynamic parameters used in the statement; similarly, the application may also know in detail the characteristics of the values that will be returned by execution of the statement. However, in other cases, the application may not know this information to the required level of detail; it is possible in some cases for the application to ascertain the information from the Information Schema, but in other cases (*e.g.*, when a returned value is derived from a computation instead of simply from a column in a table, or when dynamic parameters are supplied) this information is not generally available except in the context of preparing the statement for execution.

To provide the necessary information to applications, SQL permits an application to request the database system to *describe* a prepared statement. The description of a statement identifies the number of dynamic parameters (*describe input*) and their data type information or it identifies the number of values to be returned (*describe output*) and their data type information. The description of a statement is placed into the SQL descriptor areas already mentioned.

Many, but not all, SQL-statements can be prepared and executed dynamically.

Note: The complete list of statements that may be dynamically prepared and executed is defined in Subclause 4.22.4, "Preparable and immediately executable SQL-statements".

Certain "set statements" (<set catalog statement>, <set schema statement>, and <set names statement>) have no effect other than to set up default information (<catalog name>, <schema name>, and <character set>, respectively) to be applied to other SQL-statements that are prepared or executed immediately or that are invoked directly.

Syntax errors and Access Rule violations caused by the preparation or immediate execution of <preparable statement>s are identified when the statement is prepared (by <prepare statement>) or when it is executed (by <execute statement> or <execute immediate statement>); such violations are indicated by the raising of an exception condition.

4.25 Direct invocation of SQL

Direct invocation of SQL is a mechanism for executing direct SQL-statements, known as <direct SQL statement>s. In direct invocation of SQL, the method of invoking <direct SQL statement>s, the method of raising conditions that result from the execution of <direct SQL statement>s, the method of accessing the diagnostics information that results from the execution of <direct SQL statement>s, and the method of returning the results are implementation-defined.

4.26 Privileges

A privilege authorizes a given category of <action> to be performed on a specified base table, view, column, domain, character set, collation, or translation 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

— REFERENCES

— REFERENCES (<column name list>)

— USAGE

An <authorization identifier> is specified for each <schema definition> and <module> as well as for each SQL-session.

4.26 Privileges

A schema that is owned by a given <authorization identifier> may contain privilege descriptors that describe privileges granted to other <authorization identifier>s (grantees). The granted privileges apply to objects defined in the current schema. The WITH GRANT OPTION clause of a <grant statement> specifies whether the 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 <procedure>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 <procedure> 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 <module 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. For direct SQL, the SQL-session <authorization identifier> is always the current <authorization identifier>.

Each privilege is represented by a privilege descriptor. A privilege descriptor contains:

— the identification of the table, column, domain, character set, collation, or translation 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, 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 column in the table identified by the privilege descriptor.

Note: In this American Standard, a SELECT column privilege cannot be explicitly granted or revoked. However, for the sake of compatibility with planned future language extensions, SELECT column privilege descriptors will appear in the Information Schema.

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, character set, collation, or translation identified by the privilege descriptor.

A grantable privilege is a privilege associated with a schema that may be granted by a <grant statement>.

The phrase *applicable privileges* refers to the privileges defined by the privilege descriptors that define privileges granted to the current <authorization identifier>.

The set of applicable privileges for the current <authorization identifier> consists of the privileges defined by the privilege descriptors associated with that <authorization identifier> and the privileges defined by the privilege descriptors associated with PUBLIC.

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.

4.27 SQL-agents

An SQL-agent is an implementation-dependent entity that causes the execution of SQL-statements.

4.28 SQL-transactions

An *SQL-transaction* (sometimes simply called a "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 or by the direct invocation of SQL.

It is implementation-defined whether or not the non-dynamic or dynamic execution of an SQL-data statement or the execution of an <SQL dynamic data statement> is permitted to occur within the same SQL-transaction as the non-dynamic or dynamic execution of an SQL-schema statement. If it does occur, then the effect on any open cursor, prepared dynamic statement, 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> or direct invocation of SQL 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 <procedure> is called that results in the execution of a *transaction-initiating* SQL-statement or by direct invocation of SQL that results in the execution of a transaction-initiating <direct 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 has a *constraint mode* for each integrity constraint. The constraint mode for an integrity constraint in an SQL-transaction is described in Subclause 4.10, "Integrity constraints".

4.28 SQL-transactions

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.

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 a SQL-transaction 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 9, "SQL-transaction isolation levels and the three phenomena" specifies the phenomena that are possible and not possible for a given isolation level.

Table 9—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 11.8, "<referential constraint definition>", and in the General Rules for Subclause 12.3, "<procedure>". 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-environment being just one instance of such a manager. In such environments, an encompassing transaction shall be terminated via that other agent, which in turn interacts with the SQL-environment via an interface that may be

different from SQL (COMMIT or ROLLBACK), in order to coordinate 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-environment is beyond the scope of this American Standard. However, it is important to note that the semantics of an SQL-transaction remain as defined in the following sense:

— When an agent that is different from the SQL-agent requests the SQL-environment to roll-back an SQL-transaction, the General Rules of Subclause 14.4, "<rollback statement>", are performed.

— When such an agent requests the SQL-environment to commit an SQL-transaction, the General Rules of Subclause 14.3, "<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 14.3, "<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.

4.29 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 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: *transaction rollback—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 <procedure> 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 <procedure> within the last active <module>, or by the last execution of a <direct SQL statement> through the direct invocation of SQL. The mechanism and rules by which an SQL-environment determines whether a call to a <procedure> is the last call within the last active <module> or the last execution of a <direct SQL statement> through the direct invocation of SQL are implementation-defined.

An implementation shall 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>.

4.30 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 or by the direct invocation of SQL.

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.

An SQL-session has an SQL-session <module> that is different from any other <module> that exists simultaneously in the SQL-environment. The SQL-session <module> contains the global prepared SQL-statements that belong to the SQL-session. The SQL-session <module> contains a <module authorization clause> that specifies SCHEMA <schema name>, where the value of <schema name> is implementation-dependent.

Within an SQL-session, declared local temporary tables are effectively created by <temporary table declaration>s. Declared local temporary tables are accessible only to invocations of <procedure>s in the <module> in which they are created. The definitions of declared local temporary tables persist until the end of the SQL-session.

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 <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 catalog name that is used to effectively qualify unqualified <schema name>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The default catalog name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set catalog statement> or <set schema statement>.

4.30 SQL-sessions

An SQL-session has a default unqualified schema name that is used to effectively qualify unqualified <qualified name>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The default unqualified schema name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set schema statement>.

An SQL-session has a default character set name that is used to identify the character set implicit for <identifier>s and <character string literal>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly. The default character set name is initially set to an implementation-defined value but can subsequently be changed by the successful execution of a <set names 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 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 SQL-session <module>,

— the current default catalog name,

— the current default unqualified schema name,

— the current character set name substitution value,

— 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 contents of all SQL dynamic descriptor areas,

— the current transaction isolation level, and

— the current transaction diagnostics area limit.

4.31 Client-server operation

Within an SQL-environment, an SQL-implementation may be considered to effectively contain an SQL-client component and one or more SQL-server components.

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 <procedure> 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.

<module>s associated with the SQL-agent exist in the SQL-environment containing the SQL-client associated with the SQL-agent.

Called <procedure>s (and, analogously, <direct SQL statement>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 <module>s associated with the SQL-agent are effectively materialized with an implementation-dependent <module name> in the SQL-server. Other called <procedure>s and <direct SQL statement>s are processed by the SQL-server.

A call by the SQL-agent to a <procedure> 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.

4.32 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.

4.33 Leveling

Three levels of conformance are specified in this American Standard.

Entry SQL includes the statements for defining schemas, data manipulation language, referential integrity, check constraints, and default clause from ANSI X3.135-1989, and options for module language and embedded SQL interfaces to seven different programming languages, as well as direct execution of the data manipulation statements. It also includes features related to deprecated features from ANSI X3.135-1989 (commas and parentheses in parameter lists, the SQLSTATE parameter, and renaming columns in the <select list>), features related to incompatibilities with ANSI X3.135-1989 (colons preceding <parameter name>s, WITH CHECK OPTION constraint clarifications), and aids for transitioning from ANSI X3.135-1989 to this American Standard (<delimited identifier>s). Finally, it contains changes to correct defects found in ANSI X3.135-1989 (see Annex F, "Maintenance and interpretation of SQL").

Intermediate SQL includes major new facilities such as statements for changing schemas, dynamic SQL, and isolation levels for SQL-transactions. 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 and named constraints. 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.

4.34 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 23.3, "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 American 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 SQL 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 4.33, "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 embedded language character>

<SQL embedded language character> ::=
        <left bracket>
      | <right bracket>

<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>
      | <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> ::= ]

<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 Intermediate SQL:

   None.

2) 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>

<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>

<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>
        | <left bracket>
        | <right bracket>
```

```
<not equals operator> ::= <>

<greater than or equals operator> ::= >=

<less than or equals operator> ::= <=

<concatenation operator> ::= ||

<double period> ::= ..

<separator> ::= { <comment> | <space> | <newline> }...

<comment> ::=
       <comment introducer> [ <comment character>... ] <newline>

<comment character> ::=
         <nonquote character>
       | <quote>

<comment introducer> ::= <minus sign><minus sign>[<minus sign>...]

<newline> ::= !! implementation-defined end-of-line indicator

<key word> ::=
         <reserved word>
       | <non-reserved word>

<non-reserved word> ::=
         ADA
       | C | CATALOG_NAME | CHARACTER_SET_CATALOG | CHARACTER_SET_NAME
       | CHARACTER_SET_SCHEMA | CLASS_ORIGIN | COBOL | COLLATION_CATALOG
       | COLLATION_NAME | COLLATION_SCHEMA | COLUMN_NAME | COMMAND_FUNCTION | COMMITTED
       | CONDITION_NUMBER | CONNECTION_NAME | CONSTRAINT_CATALOG | CONSTRAINT_NAME
       | CONSTRAINT_SCHEMA | CURSOR_NAME
       | DATA | DATETIME_INTERVAL_CODE | DATETIME_INTERVAL_PRECISION | DYNAMIC_FUNCTION
       | FORTRAN
       | LENGTH
       | MESSAGE_LENGTH | MESSAGE_OCTET_LENGTH | MESSAGE_TEXT | MORE | MUMPS
       | NAME | NULLABLE | NUMBER
       | PASCAL | PLI
       | REPEATABLE | RETURNED_LENGTH | RETURNED_OCTET_LENGTH | RETURNED_SQLSTATE
       | ROW_COUNT
       | SCALE | SCHEMA_NAME | SERIALIZABLE | SERVER_NAME | SUBCLASS_ORIGIN
       | TABLE_NAME | TYPE
       | UNCOMMITTED | UNNAMED

<reserved word> ::=
         ABSOLUTE | ACTION | ADD | ALL | ALLOCATE | ALTER | AND | ANY | ARE | AS | ASC
       | ASSERTION | AT | AUTHORIZATION | AVG
       | BEGIN | BETWEEN | BIT | BIT_LENGTH | BOTH | BY
       | CASCADE | CASCADED | CASE | CAST | CATALOG | CHAR | CHARACTER | CHAR_LENGTH
       | CHARACTER_LENGTH | CHECK | CLOSE | COALESCE | COLLATE | COLLATION
       | COLUMN | COMMIT | CONNECT | CONNECTION | CONSTRAINT | CONSTRAINTS | CONTINUE
       | CONVERT | CORRESPONDING | COUNT | CREATE | CROSS | CURRENT
       | CURRENT_DATE | CURRENT_TIME | CURRENT_TIMESTAMP | CURRENT_USER | CURSOR
       | DATE | DAY | DEALLOCATE | DEC | DECIMAL | DECLARE | DEFAULT | DEFERRABLE
       | DEFERRED | DELETE | DESC | DESCRIBE | DESCRIPTOR | DIAGNOSTICS
       | DISCONNECT | DISTINCT | DOMAIN | DOUBLE | DROP
       | ELSE | END | END-EXEC | ESCAPE | EXCEPT | EXCEPTION | EXEC | EXECUTE | EXISTS
       | EXTERNAL | EXTRACT
       | FALSE | FETCH | FIRST | FLOAT | FOR | FOREIGN | FOUND | FROM | FULL
```

```
| GET | GLOBAL | GO | GOTO | GRANT | GROUP
| HAVING | HOUR
| IDENTITY | IMMEDIATE | IN | INDICATOR | INITIALLY | INNER | INPUT
| INSENSITIVE | INSERT | INT | INTEGER | INTERSECT | INTERVAL | INTO | IS
| ISOLATION
| JOIN
| KEY
| LANGUAGE | LAST | LEADING | LEFT | LEVEL | LIKE | LOCAL | LOWER
| MATCH | MAX | MIN | MINUTE | MODULE | MONTH
| NAMES | NATIONAL | NATURAL | NCHAR | NEXT | NO | NOT | NULL
| NULLIF | NUMERIC
| OCTET_LENGTH | OF | ON | ONLY | OPEN | OPTION | OR | ORDER | OUTER
| OUTPUT | OVERLAPS
| PAD | PARTIAL | POSITION | PRECISION | PREPARE | PRESERVE | PRIMARY
| PRIOR | PRIVILEGES | PROCEDURE | PUBLIC
| READ | REAL | REFERENCES | RELATIVE | RESTRICT | REVOKE | RIGHT
| ROLLBACK | ROWS
| SCHEMA | SCROLL | SECOND | SECTION | SELECT | SESSION | SESSION_USER | SET
| SIZE | SMALLINT | SOME | SPACE | SQL | SQLCODE | SQLERROR | SQLSTATE
| SUBSTRING | SUM | SYSTEM_USER
| TABLE | TEMPORARY | THEN | TIME | TIMESTAMP | TIMEZONE_HOUR | TIMEZONE_MINUTE
| TO | TRAILING | TRANSACTION | TRANSLATE | TRANSLATION | TRIM | TRUE
| UNION | UNIQUE | UNKNOWN | UPDATE | UPPER | USAGE | USER | USING
| VALUE | VALUES | VARCHAR | VARYING | VIEW
| WHEN | WHENEVER | WHERE | WITH | WORK | WRITE
| YEAR
| ZONE
```

Note: The list of <reserved word>s is considerably longer than the analogous list of <key word>s in X3.135-1989. To assist users of this American 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: AFTER, ALIAS, ASYNC, BEFORE, BOOLEAN, BREADTH, COMPLETION, CALL, CYCLE, DATA, DEPTH, DICTIONARY, EACH, ELSEIF, EQUALS, GENERAL, IF, IGNORE, LEAVE, LESS, LIMIT, LOOP, MODIFY, NEW, NONE, OBJECT, OFF, OID, OLD, OPERATION, OPERATORS, OTHERS, PARAMETERS, PENDANT, PREORDER, PRIVATE, PROTECTED, RECURSIVE, REF, REFERENCING, REPLACE, RESIGNAL, RETURN, RETURNS, ROLE, ROUTINE, ROW, SAVEPOINT, SEARCH, SENSITIVE, SEQUENCE, SIGNAL, SIMILAR, SQLEXCEPTION, SQLWARNING, STRUCTURE, TEST, THERE, TRIGGER, TYPE, UNDER, VARIABLE, VIRTUAL, VISIBLE, WAIT, WHILE, and WITHOUT.

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>. 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) SQL text containing one or more instances of <comment> is equivalent to the same SQL text with the <comment> replaced with <newline>.

8) 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.

9) The <delimited identifier body> of a <delimited identifier> shall not comprise more than 128 <delimited identifier part>s.

10) 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.

11) The <identifier body> of a <regular identifier> (with every letter that is a lower-case letter replaced by the equivalent 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 equivalent 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 American Standard or revisions thereto shall end with an <underscore>.

12) 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>".

13) 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 equivalent 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>".

14) Two <delimited identifier>s are equivalent if their <delimited identifier body>s (with all occurrences of <quote> replaced by <quote symbol> and 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>".

15) 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>.

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) No <identifier body> shall end in an <underscore>.

2) 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>
      | <datetime literal>
      | <interval 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>...

<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> <date value> <quote>

<time string> ::=
        <quote> <time value> [ <time zone interval> ] <quote>

<timestamp string> ::=
        <quote> <date value> <space> <time value> [ <time zone interval> ] <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> { <year-month literal> | <day-time literal> } <quote>

<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>
```

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>, 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>,

   c) If the <character string literal> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default character set name for the SQL-session.

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 data type of a <date literal> is DATE.

16) 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.

17) 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.

18) 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.

19) Let *datetime component* be either <years value>, <months value>, <days value>, <hours value>, <minutes value>, or <seconds value>.

20) 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>.

21) 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.

22) 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.

23) Within the definition of an <interval literal>, the <datetime value>s are constrained by the natural rules for intervals according to the Gregorian calendar.

24) 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.

25) Within the definition of a <datetime literal>, the value of the <time zone interval> shall be in the range −12:59 to +13:00.

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.

6) 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.

7) 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>.

8) 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   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>.

2) 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> ::=
         <qualified name>
       | <qualified local table name>

<qualified local table name> ::=
       MODULE <period> <local table name>

<local table name> ::= <qualified identifier>

<domain name> ::= <qualified name>

<schema name> ::=
       [ <catalog name> <period> ] <unqualified schema name>

<unqualified schema name> ::= <identifier>

<catalog name> ::= <identifier>

<qualified name> ::=
       [ <schema name> <period> ] <qualified identifier>

<qualified identifier> ::= <identifier>

<column name> ::= <identifier>

<correlation name> ::= <identifier>

<module name> ::= <identifier>

<cursor name> ::= <identifier>
```

```
<procedure name> ::= <identifier>

<SQL statement name> ::=
          <statement name>
        | <extended statement name>

<statement name> ::= <identifier>

<extended statement name> ::=
        [ <scope option> ] <simple value specification>

<dynamic cursor name> ::=
          <cursor name>
        | <extended cursor name>

<extended cursor name> ::=
        [ <scope option> ] <simple value specification>

<descriptor name> ::=
        [ <scope option> ] <simple value specification>

<scope option> ::=
          GLOBAL
        | LOCAL

<parameter name> ::= <colon> <identifier>

<constraint name> ::= <qualified name>

<collation name> ::= <qualified name>

<character set name> ::= [ <schema name> <period> ] <SQL language identifier>

<translation name> ::= <qualified name>

<form-of-use conversion name> ::= <qualified name>

<connection name> ::= <simple value specification>

<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 identified 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>,

c) If the <identifier> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default character set name for the SQL-session.

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 equivalent 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 equivalent upper-case letter), 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 American standard or revisions thereto shall end with an <underscore>.

6) If <table name> is not a <qualified local table name>, then the table identified by <table name> shall not be a declared local temporary table.

7) No <unqualified schema name> shall specify DEFINITION_SCHEMA.

8) If a <qualified name> does not contain a <schema name>, then

Case:

a) If the <qualified name> is contained in a <schema definition>, then the <schema name> that is specified or implicit in the <schema definition> is implicit.

b) If the <qualified name> is contained in a <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default <unqualified schema name> for the SQL-session is implicit.

c) Otherwise, the <schema name> that is specified or implicit for the <module> is implicit.

9) 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 <preparable statement> that is prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> or in a <direct SQL statement> that is invoked directly, then the default catalog name for the SQL-session is implicit.

d) 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.

e) Otherwise, the explicit or implicit <catalog name> contained in the <module authorization clause> is implicit.

10) Two <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.

11) 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.

12) 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 6.3, "<table reference>"). Scopes may be nested. In different scopes, the same <correlation name> may be associated with different tables or with the same table.

13) The <simple value specification> of <extended statement name> or <extended cursor name> shall not be a <literal>.

14) The data type of the <simple value specification> of <extended statement name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

15) The data type of the <simple value specification> of <extended cursor name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

16) The data type of the <simple value specification> of <descriptor name> shall be character string with an implementation-defined character set and shall have an octet length of 128 octets or less.

17) In a <descriptor name>, <extended statement name>, or <extended cursor name>, if a <scope option> is not specified, then a <scope option> of LOCAL is implicit.

18) No <authorization identifier> shall specify "PUBLIC".

19) Those <identifier>s that are valid <authorization identifier>s are implementation-defined.

20) Those <identifier>s that are valid <catalog name>s are implementation-defined.

21) If a <character set name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

22) If a <collation name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

23) If a <translation name> does not specify a <schema name>, then INFORMATION_SCHEMA is implicit.

24) 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.

25) 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.

3) A <local table name> identifies a declared local temporary table.

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) A <module name> identifies a <module>.

8) A <cursor name> identifies a cursor.

9) A <procedure name> identifies a <procedure>.

10) A <parameter name> identifies a parameter.

11) A <constraint name> identifies a table constraint, a domain constraint, or an assertion.

12) A <statement name> identifies a statement prepared by the execution of a <prepare statement>. The scope of a <statement name> is the <module> in which it appears and the current SQL-session.

13) The value of an <extended statement name> identifies a statement prepared by the execution of a <prepare statement>. If a <scope option> of GLOBAL is specified, then the scope of the <extended statement name> is the current SQL-session. If a <scope option> of LOCAL is specified or implicit, then the scope of the statement name is further restricted to the <module> in which the <extended statement name> appears.

14) A <dynamic cursor name> identifies a cursor in an <SQL dynamic statement>.

15) The value of an <extended cursor name> identifies a cursor created by the execution of an <allocate cursor statement>. If a <scope option> of GLOBAL is specified, then the scope of the <extended cursor name> is the current SQL-session. If a <scope option> of LOCAL is specified of implicit, then the scope of the cursor name is further restricted to the <module> in which the <extended cursor name> appears.

16) A <descriptor name> identifies an SQL descriptor area created by the execution of an <allocate descriptor statement>. If a <scope option> of GLOBAL is specified, then the scope of the <descriptor name> is the current SQL-session. If a <scope option> of LOCAL is specified or implicit, then the scope of the <descriptor name> is further restricted to the <module> in which the <descriptor name> appears.

17) A <catalog name> identifies a catalog.

18) A <schema name> identifies a schema.

19) A <collation name> identifies a collating sequence.

20) A <character set name> identifies a character set.

21) A <translation name> identifies a character translation.

22) A <form-of-use conversion name> identifies a form-of-use conversion. All <form-of-use conversion name>s are implementation-defined.

23) A <connection name> identifies an SQL-connection.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <extended statement name> or <extended cursor name>.

   b) 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>.

2) 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>, <SQL statement name>, <dynamic cursor name>, <constraint name>, <descriptor name>, or <character set name>.

   b) An <identifier> shall not specify a <character set specification>.

6 Scalar expressions

6.1 <data type>

Function

Specify a data type.

Format

```
<data type> ::=
        <character string type> [ CHARACTER SET <character set specification> ]
      | <national character string type>
      | <bit string type>
      | <numeric type>
      | <datetime type>
      | <interval 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>

<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>

<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>
```

```
<precision> ::= <unsigned integer>

<scale> ::= <unsigned integer>

<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>
```

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.

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 a <scale> is omitted, then a <scale> of 0 is implicit.

6) If a <precision> is omitted, then an implementation-defined <precision> is implicit.

7) CHARACTER specifies the data type character string.

8) Characters in a character string are numbered beginning with 1.

9) Case:

   a) If VARYING is not 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>.

   The maximum value of <length> is implementation-defined. <length> shall not be greater than this maximum value.

10) 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 11.21, "<domain definition>", and Subclause 11.4, "<column definition>", specify the result when <character string type> is contained in a <domain definition> or <column definition>, respectively.

11) 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".

12) BIT specifies the data type bit string.

13) Bits in a bit string are numbered beginning with 1.

14) 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.

15) The <scale> of an <exact numeric type> shall not be greater than the <precision> of the <exact numeric type>.

16) 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.

17) NUMERIC specifies the data type exact numeric, with the decimal precision and scale specified by the <precision> and <scale>.

18) 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>.

19) 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.

20) 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.

21) 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.

22) REAL specifies the data type approximate numeric, with implementation-defined precision.

23) DOUBLE PRECISION specifies the data type approximate numeric, with implementation-defined precision that is greater than the implementation-defined precision of REAL.

24) For the <approximate numeric type>s FLOAT, REAL, and DOUBLE PRECISION, the maximum and minimum values of the exponent are implementation-defined.

25) If <time precision> is not specified, then 0 is implicit. If <timestamp precision> is not specified, then 6 is implicit.

26) 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.

27) 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.

28) 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.

29) The *i*-th value of an interval data type corresponds to the *i*-th <datetime field>.

30) 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.

Access Rules

None.

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 10, "Valid values for fields in datetime items", specifies the constraints on the values of the <datetime field>s in a datetime data type.

Table 10—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) If any specification or operation attempts to cause an item of type datetime to take a value where the <datetime field> YEAR has the value greater than 9999 or less than 1, then an exception condition is raised: *data exception—invalid datetime format*.

7) 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 11, "Valid values for fields in INTERVAL items", specifies the constraints for the other <datetime field>s in the interval data type.

Table 11—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>.

8) An item of type interval can contain positive or negative intervals.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <datetime type> shall not specify a <time precision> or <timestamp precision>.

   b) A <data type> shall not be a <bit string type>.

2) 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> ::=
         <parameter specification>
       | <dynamic parameter specification>
       | <variable specification>
       | USER
       | CURRENT_USER
       | SESSION_USER
       | SYSTEM_USER
       | VALUE

<simple value specification> ::=
         <parameter name>
       | <embedded variable name>
       | <literal>

<target specification> ::=
         <parameter specification>
       | <variable specification>

<simple target specification> ::=
         <parameter name>
       | <embedded variable name>

<parameter specification> ::=
       <parameter name> [ <indicator parameter> ]

<indicator parameter> ::=
       [ INDICATOR ] <parameter name>

<dynamic parameter specification> ::= <question mark>

<variable specification> ::=
       <embedded variable name> [ <indicator variable> ]

<indicator variable> ::=
       [ INDICATOR ] <embedded variable name>
```

Syntax Rules

1) The data type of an <indicator parameter> or <indicator variable> shall be exact numeric with a scale of 0.

2) Each <parameter name> shall be contained in a <module>. Each <embedded variable name> shall be contained in an <embedded SQL statement>. Each <dynamic parameter specification> shall be contained in a <preparable statement> that is dynamically prepared in the current SQL-session through the execution of a <prepare statement> or an <execute immediate statement>.

3) If USER is specified, then CURRENT_USER is implicit.

4) The data type of CURRENT_USER, SESSION_USER, and SYSTEM_USER 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.

5) 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 definition> containing that <domain constraint>.

6) 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".

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 <parameter specification> identifies a parameter or a parameter and an indicator parameter in a <module>.

3) A <dynamic parameter specification> identifies a parameter used by a dynamically prepared statement.

4) A <variable specification> identifies a host variable or a host variable and an indicator variable.

5) A <target specification> specifies a parameter or variable that can be assigned a value.

6) If a <parameter specification> contains an <indicator parameter> and the value of the indicator parameter is negative, then the value specified by the <parameter specification> is null. Otherwise, the value specified by a <parameter specification> is the value of the parameter identified by the <parameter name>.

7) If a <variable specification> contains an <indicator variable> and the value of the indicator variable is negative, then the value specified by the <variable specification> is null. Otherwise, the value specified by a <variable specification> is the value of the variable identified by the <embedded variable name>.

8) The value specified by a <literal> is the value represented by that <literal>.

9) The value specified by CURRENT_USER is the value of the current <authorization identifier>.

10) The value specified by SESSION_USER is the value of the SQL-session <authorization identifier>.

11) 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.

12) A <simple value specification> specifies a <value specification> or <unsigned value specification> that is not null and does not have an associated <indicator parameter> or <indicator variable>.

13) 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 Intermediate SQL:

   a) In Intermediate SQL, the specific data type of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <general value specification> shall not be a <dynamic parameter specification>.

   b) A <general value specification> shall not specify VALUE.

   c) 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 <table reference>

Function

Reference a table.

Format

```
<table reference> ::=
         <table 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>

<derived column list> ::= <column name list>

<column name list> ::=
       <column name> [ { <comma> <column name> }... ]
```

Syntax Rules

1) A <correlation name> immediately contained in a <table reference> *TR* is *exposed* by *TR*. A <table 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> *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 clause of the exposed <correlation name> or exposed <table 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) 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 <table name> of *TR* is the <join condition> of *SC* and of all <joined table>s contained in *SC* that contain *TR*.

3) A <table name> that is exposed by a <table reference> *TR* shall not be the same as any other <table 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 <table name> that is exposed by a <table reference> with the same scope clause as *TR*.

5) A <table name> immediately contained in a <table reference> *TR* has a scope clause and scope defined by that <table reference> if and only if the <table 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 <table 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 <table name> is the *i*-th <column name> in that <derived column list>.

8) A <derived table> is an *updatable derived table* if and only if the <query expression> simply contained in the <subquery> of the <table subquery> of the <derived table> is updatable.

Access Rules

1) Let *T* be the table identified by the <table name> immediately contained in <table reference>. If the <table reference> is contained in any of:

   a) a <query expression> simply contained in a <cursor specification>, a <view definition>, a <direct select statement: multiple rows>, 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 *T*.

General Rules

1) The <correlation name> or exposed <table name> contained in a <table reference> defines that <correlation name> or <table name> to be an identifier of the table identified by the <table name> or <derived table> of that <table reference>.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <table reference> shall not be a <derived table>.

2) 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.

6.4 <column reference>

Function

Reference a column.

Format

```
<column reference> ::= [ <qualifier> <period> ] <column name>

<qualifier> ::=
         <table name>
       | <correlation name>
```

Syntax Rules

1) Let *CR* be the <column reference>, let *CN* be the <column name> contained in *CR*, and let *C* be the column identified by *CN*.

2) If *CR* contains a <qualifier> *Q*, then *CR* 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*.

   a) *T* shall include a column whose <column name> is *CN*.

   b) If *T* is a <table reference> in a <joined table> *J*, then *CN* shall not be a common column name in *J*.

      Note: *Common column name* is defined in Subclause 7.5, "<joined table>".

3) If *CR* does not contain a <qualifier>, then *CR* shall be contained within the scope of one or more <table name>s or <correlation name>s whose associated tables include a column whose <column name> is *CN*. Let the phrase *possible qualifiers* denote those <table name>s and <correlation name>s.

   a) Case:

      i) If the most local scope contains exactly one possible qualifier, then the qualifier *Q* equivalent to that unique <table name> or <correlation name> is implicit.

      ii) If there is more than one possible qualifier with most local scope, then:

         1) Each possible qualifier shall be a <table name> or a <correlation name> of a <table reference> that is directly contained in a <joined table> *J*.

         2) *CN* shall be a common column name in *J*.

            Note: *Common column name* is defined in Subclause 7.5, "<joined table>".

         3) The implicit qualifier *Q* is implementation-dependent. The scope of *Q* is that which *Q* would have had if *J* had been replaced by the <table reference>:

            (*J*) AS *Q*

   b) Let *T* be the table associated with *Q*.

4) The data type of *CR* is the data type of column *C* of *T*. *CN* shall uniquely identify a column of *T*.

5) If the data type of *CR* is character string, then *CR* has the *Implicit* coercibility attribute and its collating sequence is the default collating sequence for column *C* of *T*.

6) If the data type of *CR* is TIME or TIMESTAMP, then the implicit time zone of the data is the current default time zone for the SQL-session.

7) If the data type of *CR* 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 *CR*.

8) If *CR* is contained in a <table expression> *TE* and the scope clause of the <table reference> immediately containing the <table name> or <correlation name> *Q* also contains *TE*, then *CR* is an *outer reference* to the table associated with *Q*.

9) Let *CR* be the <column reference> and let *C* be the column identified by *CR*. *C* is an underlying column of *CR*. If *C* is a <derived column>, then every underlying column of *C* is an underlying column of *CR*.

Note: The underlying columns of a <derived column> are defined in Subclause 7.9, "<query specification>".

Access Rules

1) The applicable privileges shall include SELECT for *T* if *CR* is contained in any of:

   a) a <search condition> immediately contained in a <delete statement: searched> or an <update statement: searched>; or

   b) a <value expression> immediately contained in an <update source>.

General Rules

1) The <column reference> *Q.CN* references column *C* in a given row of *T*.

2) If the data type of *CR* 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 *CR*. The value of *CR*, normalized to UTC, is effectively computed as

   CR + *TZ*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL;

   None.

2) The following restrictions apply for Entry SQL;

   None.

6.5 <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 specified in Subclause 7.8, "<having clause>", and Subclause 7.9, "<query specification>".

   Note: *argument source* is defined in Subclause 7.8, "<having clause>".

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 6.4, "<column 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 <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 6.4, "<column reference>".

6) Let *DT* be the data type of the <value expression>.

7) If COUNT is specified, then the data type of the result is exact numeric with implementation-defined precision and scale of 0.

8) If MAX or MIN is specified, then the data type of the result is *DT*.

9) If SUM or AVG is specified, then:

   a) *DT* shall not be character string, bit string, 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*.

10) 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".

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*. 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 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 Intermediate SQL:

   a) If a <general set function> specifies DISTINCT, then the <value expression> shall be a <column reference>.

2) 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>.

6.6 <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> ::=
      POSITION <left paren> <character value expression>
          IN <character 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 <position expression> is specified, then the character repertoires of the two <character value expression>s shall be the same.

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.

Access Rules

None.

General Rules

1) If <position expression> is specified and neither <character value expression> is the null value, then

Case:

a) If the first <character value expression> has a length of 0, then the result is 1.

b) If the value of the first <character value expression> is equal to an identical-length substring of contiguous characters from the value of the second <character value expression>, then the result is 1 greater than the number of characters within the value of the second <character value expression> preceding the start of the first such substring.

c) Otherwise, the result is 0.

2) If <position expression> is specified and either <character value expression> is the null value, then the result is the null value.

3) If <extract expression> is specified, then

Case:

a) If the value of the <interval value expression> or the <datetime value expression> is not the null value, then

Case:

i) 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.

ii) 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*)

b) Otherwise, the result is the null value.

4) If a <char length expression> is specified, then

Case:

a) Let *S* be the <string value expression>. If the value of *S* is not the null value, then

Case:

i) If the data type of *S* is a character data type, then the result is the number of characters in the value of *S*.

ii) Otherwise, the result is OCTET_LENGTH(*S*).

b) Otherwise, the result is the null value.

5) If an <octet length expression> is specified, then

Case:

a) Let *S* be the <string value expression>. If the value of *S* is not the null value, then the result is the smallest integer not less than the quotient of the division (BIT_LENGTH(*S*)/8).

b) Otherwise, the result is the null value.

6) If a <bit length expression> is specified, then

Case:

a) Let *S* be the <string value expression>. If the value of *S* is not the null value, then the result is the number of bits in the value of *S*.

b) Otherwise, the result is the null value.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   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>.

2) 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>.

6.7 <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>

<character value function> ::=
        <character substring function>
      | <fold>
      | <form-of-use conversion>
      | <character translation>
      | <trim function>

<character substring function> ::=
      SUBSTRING <left paren> <character value expression> FROM <start position>
                  [ FOR <string length> ] <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>

<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> 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 <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.

4) 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*.

5) 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*.

6) 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.

7) 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>.

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 the null value, then the result of the <character substring function> is the 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 <fold> is specified, then:

   a) Let *S* be the value of the <character value expression>.

   b) If *S* is the null value, then the result of the <fold> is the 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>.

3) If a <character translation> is specified, then

   Case:

   a) If the value of <character value expression> is the null value, then the result of the <character translation> is the 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.

4) If a <form-of-use conversion> is specified, then

   Case:

   a) If the value of <character value expression> is the null value, then the result of the <form-of-use conversion> is the 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>.

5) 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 the null value, then the result of the <trim function> is the 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.

6) 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 the null value, then the result of the <bit substring function> is the 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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

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>.

2) 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>.

6.8 <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>.

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 generally contains more than one reference to one or more <datetime value function>s, then all such references are effectively evaluated 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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <time precision> or <timestamp precision>.

2) 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>.

6.9 <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", to the data types of all <result expression>s in the <case specification>.

Access Rules

None.

General Rules

1) Case:

   a) If a <result> specifies NULL, then its value is the 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 Intermediate SQL:

   None.

2) 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>.

6.10 <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>, 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
EN	Y	Y	Y	Y	N	N	N	N	N	M	M
AN	Y	Y	Y	Y	N	N	N	N	N	N	N
C	Y	Y	M	M	Y	Y	Y	Y	Y	Y	Y
B	N	N	Y	Y	Y	Y	N	N	N	N	N
D	N	N	Y	Y	N	N	Y	N	Y	N	N
T	N	N	Y	Y	N	N	N	Y	Y	N	N
TS	N	N	Y	Y	N	N	Y	Y	Y	N	N
YM	M	N	Y	Y	N	N	N	N	N	Y	N
DT	M	N	Y	Y	N	N	N	N	N	N	Y

Where:

EN = Exact Numeric
AN = Approximate Numeric
C = Character (Fixed- or Variable-length)

```
FC = Fixed-length Character
VC = Variable-length Character
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
```

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 or variable-length character string, then the character repertoires of *SD* and *TD* shall be the same.

8) If *TD* is a fixed-length or variable-length 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.

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 or if *SV* is the null value, then the result of the <cast specification> is the null value.

   b) Otherwise, let *TV* be the result of the <cast specification> as specified in the remaining General Rules of this Subclause.

3) 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*.

4) 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*)

5) 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 or variable-length 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 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*.

6) If *TD* is variable-length 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 or variable-length 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 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*.

7) 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*.

8) 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*.

9) 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> 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, an exception condition is raised: *data exception—invalid character value for cast*.

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*.

10) 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> 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, an exception condition is raised: *data exception—invalid character value for cast*.

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.

11) 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> 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, an exception condition is raised: *data exception—invalid character value for cast*.

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*.

12) 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> 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, an exception condition is raised: *data exception—invalid character value for cast*.

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*.

13) 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 Intermediate SQL:

   None.

2) 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>.

6.11 <value expression>

Function

Specify a value.

Format

```
<value expression> ::=
        <numeric value expression>
      | <string value expression>
      | <datetime value expression>
      | <interval value expression>

<value expression primary> ::=
        <unsigned value specification>
      | <column reference>
      | <set function specification>
      | <scalar subquery>
      | <case expression>
      | <left paren> <value expression> <right paren>
      | <cast specification>
```

Syntax Rules

1) The data type of a <value expression> is the data type of the <numeric value expression>, <string value expression>, <datetime value expression>, or <interval value expression>, respectively.

2) 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>, <column reference>, <set function specification>, <scalar subquery>, <case expression>, <value expression>, or <cast specification> immediately contained in the <value expression primary>.

3) 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*.

Access Rules

None.

General Rules

1) When a <value expression> *V* is evaluated for a row of a table, each reference to a column of that table by a <column reference> directly contained in *V* is a reference to the value of that column in that row.

2) 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 null value.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

6.12 <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) 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.

2) 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.

3) The data type of a <factor> is that of the immediately contained <numeric primary>.

4) The data type of a <numeric primary> shall be numeric.

Access Rules

None.

General Rules

1) If the value of any <numeric primary> simply contained in a <numeric value expression> is the null value, then the result of the <numeric value expression> is the null value.

2) If the <numeric value expression> contains only a <numeric primary>, then the value 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 Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

6.13 <string value expression>

Function

Specify a character string value or a bit string value.

Format

```
<string value expression> ::=
        <character value expression>
      | <bit 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>

<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) The data type of a <character primary> shall be character string.

2) 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.

3) 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>.

4) Case:

a) If <character factor> is specified, then

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 6.7, "<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".

5) The data type of a <bit primary> shall be bit string.

6) 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*.

b) Otherwise, the data type of a <bit value expression> is the data type of the <bit factor>.

Access Rules

None.

General Rules

1) If the value of any <character primary> simply contained in a <character value expression> is the null value, then the result of the <character value expression> is the null value.

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 the null value, then the result of the <concatenation> is the 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 the value of any <bit primary> simply contained in a <bit value expression> is the null value, then the result of the <bit value expression> is the null value.

4) 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 the null value, then the result of the <bit concatenation> is the 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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

   b) Conforming Intermediate SQL language shall contain no <bit value expression>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <character value expression> shall not be a <concatenation>.

6.14 <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 value expression>
```

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.

Access Rules

None.

General Rules

1) If the result of any <datetime primary>, <interval value expression>, <datetime value expression>, or <interval term> simply contained in a <datetime value expression> is the null value, then the result of the <datetime value expression> is the null value.

2) If <time zone> is specified or implied and the <interval value expression> immediately contained in <time zone specifier> is the null value, then the result of the <datetime value expression> is the null value.

3) If a <datetime value expression> immediately contains the operator + or −, then the result is effectively evaluated as follows:

   a) Case:

      i) If <datetime value expression> immediately contains the operator + and the <interval value expression> or <interval term> is not negative, or if <datetime value expression> immediately contains the operator − 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 10, "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 value expression> 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 Intermediate SQL:

None.

2) 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>.

6.15 <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) An <interval primary> shall specify <interval qualifier> only if the <interval primary> specifies a <dynamic parameter specification>.

3) Case:

   a) If the <interval value expression> simply contains an <interval qualifier>, then the result contains 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>.

4) 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.

5) If <datetime value expression> is specified, then <datetime value expression> and <datetime term> shall be comparable.

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 result of any <interval primary>, <datetime value expression>, <datetime term>, or <factor> simply contained in an <interval value expression> is the null value, then the result of the <interval value expression> is the null value.

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 Intermediate SQL:

None.

2) 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>.

7 Query expressions

7.1 <row value constructor>

Function

Specify an ordered set of values to be constructed into a row or partial row.

Format

```
<row value constructor> ::=
         <row value constructor element>
       | <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

<default specification> ::=
       DEFAULT
```

Syntax Rules

1) If a <row value constructor> simply contains a <null specification> or a <default specification>, then the <row value constructor> shall be simply contained in a <query expression> that is simply contained in an <insert statement>.

2) 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>".

3) A <row value constructor> that immediately contains a <row value constructor element> *X* is equivalent to a <row value constructor> of the form

   (*X*)

4) The data types of the column or columns of a <row value constructor> are the data types of the <row value constructor element> or <row value constructor element>s or the columns of the <row subquery> simply contained in the <row value constructor>.

5) If a <row value constructor> is derived from a <row subquery>, then the degree of the <row value constructor> is the degree of the table resulting from the <row subquery>; otherwise, the degree of the <row value constructor> is the number of <row value constructor element>s that occur in its specification.

Access Rules

None.

General Rules

1) The value of a <null specification> is the null value.

2) The value of a <default specification> is 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>.

3) Case:

   a) If a <row value constructor list> is specified, then the result of the <row value constructor> is a row of columns whose *i*-th column has an implementation-dependent name different from the <column name> of all other columns contained in the SQL-statement and whose value is the value of the *i*-th <row value constructor element> in the <row value constructor list>.

   b) 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* 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 Intermediate SQL:

   a) A <row value constructor> that is not simply contained in a <table value constructor> shall not contain more than one <row value constructor element>.

2) 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 <table value constructor>

Function

Specify a set of <row value constructor>s to be constructed into a table.

Format

```
<table value constructor> ::=
       VALUES <table value constructor list>

<table value constructor list> ::=
       <row value constructor> [ { <comma> <row value constructor> }... ]
```

Syntax Rules

1) All <row value constructor>s shall be of the same degree.

Access Rules

None.

General Rules

1) 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 constructor> 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 constructor>.

2) The result of the <table value constructor> is the same as the result of

   T_1 [UNION ALL T_2 [. . . UNION ALL T_n] . . .]

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   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>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.3 <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 in which the descriptor of the *i*-th column is the same as the descriptor of the *i*-th column of the table specified by the <from clause>.

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 Intermediate SQL:

   None.

2) 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>.

7.4 <from clause>

Function

Specify a table derived from one or more named tables.

Format

```
<from clause> ::= FROM <table reference> [ { <comma> <table reference> }... ]
```

Syntax Rules

1) Case:

   a) If the <from clause> contains a single <table reference> with no intervening <derived table> or <joined table>, then the descriptor of the result of the <from clause> is the same as the descriptor of the table identified by that <table reference>.

   b) If the <from clause> 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 <from clause> 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 <from clause> and in the order in which the columns are defined within each table.

Access Rules

None.

General Rules

1) Case:

   a) If the <from clause> contains a single <table reference> with no intervening <derived table> or <joined table>, then the result of the <from clause> is the table identified by that <table reference>.

   b) If the <from clause> contains more than one <table reference> with no intervening <derived table> or <joined table>, then the result of the <from clause> 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 <from clause>.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) If the table identified by <table name> is a grouped view, then the <from clause> shall contain exactly one <table reference>.

7.5 <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>

<join condition> ::= ON <search condition>

<named columns join> ::=
      USING <left paren> <join column list> <right paren>

<join type> ::=
        INNER
      | <outer join type> [ OUTER ]
      | UNION

<outer join type> ::=
        LEFT
      | RIGHT
      | FULL

<join column list> ::= <column name list>
```

Syntax Rules

1) 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

2) 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.

3) If a <qualified join> is specified and a <join type> is not specified, then INNER is implicit.

4) 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 6.4, "<column reference>".

5) 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*.

6) 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

7) 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.

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_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.

9) 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.

10) The <joined table> is a read-only 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 null value. Let X_2 be U_2 extended on the left with D_1 columns containing the 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 TN
      UNION ALL
      SELECT * FROM XN_1

   c) If RIGHT is specified, then let S be the multiset of rows resulting from:

      SELECT * FROM TN
      UNION ALL
      SELECT * FROM XN_2

   d) If FULL is specified, then let S be the multiset of rows resulting from:

      SELECT * FROM TN
      UNION ALL
      SELECT * FROM XN_1
      UNION ALL
      SELECT * FROM XN_2

   e) If UNION is specified, then let S be the multiset of rows resulting from:

      SELECT * FROM XN_1
      UNION ALL
      SELECT * FROM XN_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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <cross join>.

   b) Conforming Intermediate SQL language shall not specify UNION JOIN.

2) 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>.

7.6 <where clause>

Function

Specify a table derived by the application of a <search condition> to the result of the preceding <from clause>.

Format

```
<where clause> ::= WHERE <search condition>
```

Syntax Rules

1) 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 6.4, "<column reference>".

2) 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 the <column reference> in the <set function specification> shall be an outer reference.

   Note: *Outer reference* is defined in Subclause 6.4, "<column reference>".

3) 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>.

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.

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 6.4, "<column reference>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

7.7 <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> ::=
       <column reference> [ <collate clause> ]
```

Syntax Rules

1) 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>.

2) Each <column reference> in the <group by clause> shall unambiguously reference a column of *T*. A column referenced in a <group by clause> is a grouping column.

3) 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*.

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.

2) Every row of a given group contains equal values of a given grouping column. When a <search condition> or <value expression> is applied to a group, a reference to a grouping column is a reference to that value.

   Note: See the General Rules of Subclause 8.2, "<comparison predicate>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

7.8 <having clause>

Function

Specify a grouped table derived by the elimination of groups from the result of the previously specified clause that do not meet the <search condition>.

Format

```
<having clause> ::= HAVING <search condition>
```

Syntax Rules

1) If neither a <where clause> nor a <group by clause> is specified, then let *T* be the result of the preceding <from clause>; if a <where clause> is specified, but a <group by clause> is not specified, then let *T* be the result of the preceding <where clause>; otherwise, let *T* be the result of the preceding <group by clause>. 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 6.4, "<column reference>".

2) 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>.

3) 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*.

Access Rules

None.

General Rules

1) Let *T* be the result of the preceding <from clause>, <where clause>, or <group by clause>. If that clause is not a <group by clause>, then *T* consists of a single group and does not have a grouping column.

2) The <search condition> is applied to each group of *T*. The result of the <having clause> is a grouped table of those groups of *T* for which the result of the <search condition> is true.

3) When the <search condition> is applied to a given group of *T*, 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.

   Note: *Outer reference* is defined in Subclause 6.4, "<column reference>".

4) Each <subquery> in the <search condition> is effectively executed for each group of *T* and the result used in the application of the <search condition> to the given group of *T*. If any executed <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 *T*.

Note: *Outer reference* is defined in Subclause 6.4, "<column reference>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

7.9 <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>
       | <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) The degree of the table specified by a <query specification> is equal to the cardinality of the <select list>.

3) 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* is referenced exactly once. The columns are referenced in the ascending sequence of their ordinal position within *T*.

4) The <select sublist> "<qualifier>.*" for some <qualifier> *Q* is equivalent to a <value expression> sequence in which each <value expression> is a <column reference> *CR* that references a column of *T* that is not a common column of a <joined table>. Each column of *T* that is not a common column of a <joined table> shall be referenced exactly once. The columns shall be referenced in the ascending sequence of their ordinal positions within *T*.

   Note: *common column of a <joined table>* is defined in Subclause 7.5, "<joined table>".

5) 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*.

6) 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*.

7) If *T* is a grouped table, then each <column reference> in each <value expression> that references a column of *T* shall reference a grouping column or be specified within a <set function specification>. If *T* is not a grouped table and any <value expression> contains a <set function specification> that contains a reference to a column of *T* or any <value expression> directly contains a <set function specification> that does not contain an outer reference, then every <column reference> in every <value expression> that references a column of *T* shall be specified within a <set function specification>.

8) Each column of the table that is the result of a <query specification> 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.

9) Case:

   a) If the *i*-th <derived column> in the <select list> specifies an <as clause> that contains a <column name> *C*, then the <column name> of the *i*-th column of the result is *C*.

   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 *C*.

   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 SQL-statement.

10) A column of the table that is the result of a <query specification> is *possibly nullable* if and only if it contains a <column reference> for a column *C* that is possibly nullable, an <indicator parameter>, an <indicator variable>, 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.

11) 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>.

12) A <query specification> *QS* is updatable if and only if the following conditions hold:

   a) *QS* does not specify DISTINCT.

   b) Every <value expression> contained in the <select list> immediately contained in *QS* consists of a <column reference>, and no <column reference> appears more than once.

   c) The <from clause> immediately contained in the <table expression> immediately contained in *QS* specifies exactly one <table reference> and that <table reference> refers either to a base table or to an updatable derived table.

      Note: *updatable derived table* is defined in Subclause 6.3, "<table reference>".

   d) If the <table expression> immediately contained in *QS* immediately contains a <where clause> *WC*, then no leaf generally underlying table of *QS* shall be a generally underlying table of any <query expression> contained in *WC*.

e) The <table expression> immediately contained in *QS* does not include a <group by clause> or a <having clause>.

13) 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*.

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>.

Case:

1) If the <set quantifier> DISTINCT is not specified, then the table is the result of the <query specification>.

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.

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>.

Case:

1) If the <set quantifier> DISTINCT is not specified, then the table is the result of the <query specification>.

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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) The <set quantifier> DISTINCT shall not be specified more than once in a <query specification>, excluding any <subquery> of that <query specification>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <query specification> is not 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>.

7.10 <query expression>

Function

Specify a table.

Format

```
<query expression> ::=
        <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>

<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>

<explicit table> ::= TABLE <table name>

<corresponding spec> ::=
      CORRESPONDING [ BY <left paren> <corresponding column list> <right paren> ]

<corresponding column list> ::= <column name list>
```

Syntax Rules

1) Let *T* be the table specified by the <query expression>.

2) The <explicit table>

   TABLE <table name>

   is equivalent to the <query expression>

   (SELECT * FROM <table name>)

3) Let *set operator* be UNION [ALL], EXCEPT [ALL], or INTERSECT [ALL].

4) *T* is an updatable table and the <query expression> is 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 updatable.

5) 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) Otherwise, the column descriptor of the *i*-th column of the <simple table> is 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 SQL-statement.

6) 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>.

7) 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>.

8) 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.

9) 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.

10) 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 same as the column descriptor of the *i*-th column of the <non-join query primary>.

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 of the *i*-th column of *TR* is determined by applying Subclause 9.3, "Set operation result data types", to the data types of the *i*-th column of *T1* and the *i*-th column of *T2*. If the *i*-th column 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 *T* is possibly nullable.

11) 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>.

12) 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 of the *i*-th column of *TR* is determined by applying Subclause 9.3, "Set operation result data types", to the data types of the *i*-th column of *T1* and the *i*-th column of *T2*. If the *i*-th column 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 *T* is possibly nullable.

13) 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>.

14) The *simply underlying table*s of a <query expression> are the tables identified by those <table name>s, <query expression>s, and <derived table>s contained in the <query expression> without an intervening <derived table>, an intervening <query specification>, or an intervening <join condition>.

15) 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.

16) 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*.

Access Rules

None.

General Rules

1) 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 $n \geq 0$.

ii) 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*.

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*.

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*.

B) Otherwise, *T* contains no duplicates of *R*.

iii) If ALL is specified, then

Case:

1) If UNION is specified, then the number of duplicates of *R* that *T* contains is $(m + n)$.

2) If EXCEPT is specified, then the number of duplicates of *R* that *T* contains is the maximum of ($m - n$) and 0.

3) If INTERSECT is specified, then the number of duplicates of *R* that *T* contains is the minimum of *m* and *n*.

Note: See the General Rules of Subclause 8.2, "<comparison predicate>".

2) 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*)

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

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>.

2) 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.

7.11 <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) The data type of a <scalar subquery> is the data type of the column of the <query expression> immediately contained in the <scalar subquery>.

4) The data types of the columns of a <row subquery> or <table subquery> are the data types of the respective columns of the <query expression> immediately contained in the <row subquery> or <table subquery>.

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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

2) 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>.

8 Predicates

8.1 <predicate>

Function

Specify a condition that can be evaluated to give a truth value of *true*, *false*, or *unknown*.

Format

```
<predicate> ::=
        <comparison predicate>
      | <between predicate>
      | <in predicate>
      | <like predicate>
      | <null predicate>
      | <quantified comparison predicate>
      | <exists predicate>
      | <unique predicate>
      | <match predicate>
      | <overlaps 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 8.6, "<null predicate>", Subclause 8.7, "<quantified comparison predicate>", Subclause 8.8, "<exists predicate>", Subclause 8.9, "<unique predicate>", Subclause 8.10, "<match predicate>", or Subclause 8.11, "<overlaps predicate>", as appropriate.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <predicate> shall not be a <match predicate>.

2) 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>.

8.2 <comparison predicate>

Function

Specify a comparison of two row values.

Format

```
<comparison predicate> ::=
       <row value constructor> <comp op> <row value constructor>

<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) The two <row value constructor>s shall be of the same degree.

2) Let *corresponding values* be values with the same ordinal position in the two <row value constructor>s.

3) The data types of the corresponding values of the two <row value constructor>s shall be comparable.

4) Let *X* be a value in the first <row value constructor> and *Y* be the corresponding value in the second <row value constructor>. 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.

Access Rules

None.

General Rules

1) Let *X* and *Y* be any two corresponding <row value constructor element>s. Let *XV* and *YV* be the values represented by *X* and *Y*, respectively.

   Case:

   a) If *XV* or *YV* is the null value, then "*X* <comp op> *Y*" is unknown.

   b) 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.

      ii) "*X* <> *Y*" is true if and only if *XV* and *YV* are not equal.

      iii) "*X* < *Y*" is true if and only if *XV* is less than *YV*.

iv) "*X* > *Y*" is true if and only if *XV* is greater than *YV*.

v) "*X* <= *Y*" is true if and only if *XV* is not greater than *YV*.

vi) "*X* >= *Y*" is true if and only if *XV* is not less than *YV*.

vii) "*X* <comp op> *Y*" is false if and only if "*X* <comp op> *Y*" is not true.

2) Numbers are compared with respect to their algebraic value.

3) 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 the operations MAX, MIN, DISTINCT, references to 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 10.4, "<character set specification>", and Subclause 11.28, "<character set definition>".

4) 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.

5) 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 4.5.1, "Datetimes".

6) 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)

7) 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 <row value constructor element>s 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 some *i*.

c) "$R_x < R_y$" is true if and only if $RX_i = RY_i$ for all $i < n$ and $RX_n < RY_n$ for some *n*.

d) "$R_x > R_y$" is true if and only if $RX_i = RY_i$ for all $i < n$ and $RX_n > RY_n$ for some *n*.

e) "R_x <= R_y" is true if and only if $R_x = R_y$ or $R_x < R_y$.

f) "R_x >= R_y" is true if and only if $R_x = R_y$ or $R_x > R_y$.

g) "$R_x = R_y$" is false if and only if "R_x <> R_y" is true.

h) "R_x <> R_y" is false if and only if "$R_x = R_y$" is true.

i) "$R_x < R_y$" is false if and only if "R_x >= R_y" is true.

j) "$R_x > R_y$" is false if and only if "R_x <= R_y" is true.

k) "R_x <= R_y" is false if and only if "$R_x > R_y$" is true.

l) "R_x >= R_y" is false if and only if "$R_x < R_y$" is true.

m) "R_x <comp op> R_y" is unknown if and only if "R_x <comp op> R_y" is neither true nor false.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

None.

2) 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 constructor> [ NOT ] BETWEEN
         <row value constructor> AND <row value constructor>
```

Syntax Rules

1) The three <row value constructor>s shall be of the same degree.

2) Let respective values be values with the same ordinal position in the two <row value constructor>s.

3) The data types of the respective values of the three <row value constructor>s shall be comparable.

4) Let *X*, *Y*, and *Z* be the first, second, and third <row value constructor>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*".

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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 constructor>
         [ 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) Let *IVL* be an <in value list>.

   (*IVL*)

   is equivalent to the <table value constructor>:

   (VALUES *IVL*)

2) Let *RVC* be the <row value constructor> and let *IPV* be the <in predicate value>.

3) The expression

   RVC NOT IN *IPV*

   is equivalent to

   NOT (*RVC* IN *IPV*)

4) 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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain a <value expression> in an <in value list> that is not a <value specification>.

2) 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> ::=
       <match value> [ NOT ] LIKE <pattern>
         [ ESCAPE <escape character> ]

<match value> ::= <character value expression>

<pattern> ::= <character value expression>

<escape character> ::= <character value expression>
```

Syntax Rules

1) The data types of <match value>, <pattern>, and <escape character> shall be character string. <match value>, <pattern>, and <escape character> shall be comparable.

2) Let *M* be the result of the <character value expression> of the <match value>, let *P* be the result of the <character value expression> of the <pattern>, and let *E* be the result of the <character value expression> of the <escape character> if one is specified.

3) "*M* NOT LIKE *P*" is equivalent to "NOT (*M* LIKE *P*)".

4) Case:

   a) 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 <match value> as comparand 1 and <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 <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.

Access Rules

None.

General Rules

1) If an <escape character> is specified and *M*, *P*, or *E* is the null value, then

   M LIKE *P* ESCAPE *E*

   is unknown.

2) If an <escape character> is not specified and *M* or *P* is the null value, then

   M LIKE *P*

   is unknown.

3) Case:

   a) If an <escape character> is specified, then:

      i) If the length in characters of *E* is not equal to 1, then an exception condition is raised: *data exception—invalid escape character*.

      ii) If there is not a partitioning of the string *P* into substrings such that each substring has length 1 or 2, no substring of length 1 is the escape character *E*, and each substring of length 2 is the escape character *E* followed by either the escape character *E*, 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 *P*, 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.

   b) If an <escape character> is not specified, then each <underscore> character in *P* represents an arbitrary character specifier, each <percent> character in *P* represents an arbitrary string specifier, and each character in *P* that is neither the <underscore> character nor the <percent> character represents itself.

4) The string *P* is a sequence of the minimum number of substring specifiers such that each <character representation> of *P* 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.

5) Case:

   a) If *M* and *P* are character strings whose lengths are variable and if the lengths of both *M* and *P* are 0, then

      M LIKE *P*

      is true.

   b) The <predicate>

      M LIKE *P*

      is true if there exists a partitioning of *M* into substrings such that:

i) A substring of *M* is a sequence of 0 or more contiguous <character representation>s of *M* and each <character representation> of *M* is part of exactly one substring.

ii) If the *i*-th substring specifier of *P* is an arbitrary character specifier, the *i*-th substring of *M* is any single <character representation>.

iii) If the *i*-th substring specifier of *P* is an arbitrary string specifier, then the *i*-th substring of *M* is any sequence of 0 or more <character representation>s.

iv) If the *i*-th substring specifier of *P* is neither an arbitrary character specifier nor an arbitrary string specifier, then the *i*-th substring of *M* is equal to that substring specifier according to the collating sequence of the <like predicate>, without the appending of <space> characters to *M*, and has the same length as that substring specifier.

v) The number of substrings of *M* is equal to the number of substring specifiers of *P*.

c) Otherwise,

M LIKE *P*

is false.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) The <match value> shall be a <column reference>.

b) A <pattern> shall be a <value specification>.

c) An <escape character> shall be a <value specification>.

8.6 <null predicate>

Function

Specify a test for a null value.

Format

```
<null predicate> ::= <row value constructor> IS [ NOT ] NULL
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *R* be the value of the <row value constructor>.

2) If all the values in *R* are the null value, then "*R* IS NULL" is true; otherwise, it is false.

3) If none of the values in *R* are the 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 12, "<null predicate> semantics", specifies this behavior.

Table 12—<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*

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <row value constructor> shall be a <column reference>.

8.7 <quantified comparison predicate>

Function

Specify a quantified comparison.

Format

```
<quantified comparison predicate> ::=
       <row value constructor> <comp op> <quantifier> <table subquery>

<quantifier> ::= <all> | <some>

<all> ::= ALL

<some> ::= SOME | ANY
```

Syntax Rules

1) The <row value constructor> shall be of the same degree as the result of the <table subquery>.

2) The data types of the values of the <row value constructor> shall be respectively comparable to those of the columns of the <table subquery>.

3) 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 constructor> 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 unknown.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.8 <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 Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

8.9 <unique predicate>

Function

Specify a test for the absence of duplicate rows.

Format

```
<unique predicate> ::= UNIQUE <table subquery>
```

Syntax Rules

None.

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 Intermediate SQL:

   None.

2) 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>.

8.10 <match predicate>

Function

Specify a test for matching rows.

Format

```
<match predicate> ::=
      <row value constructor> MATCH [ UNIQUE ] [ PARTIAL | FULL ] <table subquery>
```

Syntax Rules

1) The <row value constructor> shall be of the same degree as the <table subquery>.

2) The data types of the values of the <row value constructor> shall be respectively comparable to those of the corresponding columns of the <table subquery>.

3) 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 constructor>.

2) If neither PARTIAL nor FULL is specified, then

   Case:

   a) If some value in *R* is the null value, then the <match predicate> is true.

   b) If no value in *R* is the 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 the 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 the null value, then the <match predicate> is true.

b) If no values in *R* are the 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 Intermediate SQL:

a) Conforming Intermediate SQL language shall not contain any <match predicate>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

8.11 <overlaps predicate>

Function

Specify a test for an overlap between two events.

Format

```
<overlaps predicate> ::=
       <row value constructor 1> OVERLAPS <row value constructor 2>

<row value constructor 1> ::= <row value constructor>

<row value constructor 2> ::= <row value constructor>
```

Syntax Rules

1) The degree of <row value constructor 1> and <row value constructor 2> shall both be 2.

2) The data type of the first column of each <row value constructor> shall be a datetime data type and the first column of each <row value constructor> shall be comparable.

   Note: Two datetimes are comparable only if they have the same <datetime field>s; see Subclause 4.5.1, "Datetimes".

3) The data type of the second column of each <row value constructor> 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 constructor>.

   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 constructor>.

Access Rules

None.

General Rules

1) Let *D1* be the value of the first column of <row value constructor 1> and *D2* be the value of the first column of <row value constructor 2>.

2) Case:

   a) If the data type of the second column of <row value constructor 1> is a datetime data type, then let *E1* be the value of the second column of <row value constructor 1>.

   b) If the data type of the second column of <row value constructor 1> is INTERVAL, then let *I1* be the value of the second column of <row value constructor 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 constructor 2> is a datetime data type, then let *E2* be the value of the second column of <row value constructor 2>.

   b) If the data type of the second column of <row value constructor 2> is INTERVAL, then let *I2* be the value of the second column of <row value constructor 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 Intermediate SQL:

   None.

2) 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>.

8.12 <search condition>

Function

Specify a condition that has the truth value *true*, *false*, or *unknown*, depending on the result of applying boolean operators to specified conditions.

Format

```
<search condition> ::=
        <boolean term>
      | <search condition> OR <boolean term>

<boolean term> ::=
        <boolean factor>
      | <boolean term> AND <boolean factor>

<boolean factor> ::=
      [ NOT ] <boolean test>

<boolean test> ::=
      <boolean primary> [ IS [ NOT ] <truth value> ]

<truth value> ::=
        TRUE
      | FALSE
      | UNKNOWN

<boolean primary> ::=
        <predicate>
      | <left paren> <search condition> <right paren>
```

Syntax Rules

1) If NOT is specified in a <boolean test>, then let *BP* be the contained <boolean primary> and let *TV* be the contained <truth value>. The <boolean test> is equivalent to:

   (NOT (*BP* IS *TV*))

Access Rules

None.

General Rules

1) The result is derived by the application of the specified boolean operators (“AND”, “OR”, “IS”, and “NOT”) to the results derived from each <predicate> evaluation. If boolean operators are not specified, then the result of the <search condition> is the result of the specified <predicate>.

2) NOT(*true*) is false, NOT(*false*) is true, and NOT(*unknown*) is unknown. Table 13, "Truth table for the AND boolean", Table 14, "Truth table for the OR boolean", and Table 15, "Truth table for the IS boolean" specify the semantics of AND, OR, and IS, respectively.

Table 13—Truth table for the AND boolean

AND	*true*	*false*	*unknown*
true	*true*	*false*	*unknown*
false	*false*	*false*	*false*
unknown	*unknown*	*false*	*unknown*

Table 14—Truth table for the OR boolean

OR	*true*	*false*	*unknown*
true	*true*	*true*	*true*
false	*true*	*false*	*unknown*
unknown	*true*	*unknown*	*unknown*

Table 15—Truth table for the IS boolean

IS	TRUE	FALSE	UNKNOWN
true	*true*	*false*	*false*
false	*false*	*true*	*false*
unknown	*false*	*false*	*true*

3) 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <boolean test> shall not specify a <truth value>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

9 Data assignment rules

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, bit string, numeric, datetime, or interval, then the data type of *V* shall be a mutually assignable character string type, a bit string type, a numeric type, the same datetime type, or a comparable interval type, respectively.

General Rules

1) If *V* is the null value, then

   Case:

   a) If an indicator is specified for *T*, then that indicator is set to −1.

   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 not the null value and *T* has an indicator, then

   Case:

   a) If the data type of *T* is character string or bit string and the length in characters or bits, respectively, *M* of *V* is greater than the length in characters or bits, 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.

3) If *V* is not the 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 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*.

g) 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*.

h) 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*.

i) 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*.

j) 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*.

k) If the data type of *T* is numeric and there is an approximation obtained by rounding or truncation of the numerical value of *V* for the data type of *T*, then the value of *T* is set to such an approximation.

   If there is no such approximation, then an exception condition is raised: *data exception—numeric value out of range*.

   If the data type of *T* is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

l) If the data type of *T* is datetime and there is a representation of the value of *V* in the data type of *T*, then the value of *T* is set to that representation.

m) If the data type of *T* is interval and there is a representation of the value of *V* in the data type of *T*, then the value of *T* is set to that representation. Otherwise, an exception condition is raised: *data exception—interval field overflow*.

9.2 Store assignment

Function

Specify rules for value assignments that store 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, bit string, numeric, datetime, or interval, then the data type of *V* shall be character string, bit string, numeric, the same datetime type, or a comparable interval type, respectively.

General Rules

1) Let *T* be an object column.

2) If the value of *V* is the null value, then the value of *T* is set to the null value.

3) Otherwise, let *V* denote a non-null value of *T*.

   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 *M* of *V* is larger than *L*, 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*.

      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*.

   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 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.

   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 *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*.

f) 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*.

g) 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*.

h) 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*.

i) 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*.

j) 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*.

k) If the data type of *T* is numeric and there is an approximation obtained by rounding or truncation of the numerical value of *V* for the data type of *T*, then the value of *T* is set to such an approximation.

   If there is no such approximation, then an exception condition is raised: *data exception—numeric value out of range*.

   If the data type of *T* is exact numeric, then it is implementation-defined whether the approximation is obtained by rounding or by truncation.

l) If the data type of *T* is datetime and there is a representation of the value of *V* in the data type of *T*, then the value of *T* is set to that representation.

m) If the data type of *T* is interval and there is a representation of the value of *V* in the data type of *T*, then the value of *T* is set to that representation. Otherwise, an exception condition is raised: *data exception—interval field overflow*.

4) If 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

Function

Specify the Syntax Rules and result data types for <case 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 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*.

      ii) 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 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*.

   d) 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.

   e) 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.

f) 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.

General Rules

None.

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 a significance of ordering 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 be more significant than <end field>,

b) <start field> shall not specify MONTH, and

c) if <start field> specified YEAR, then <end field> shall specify 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 Intermediate SQL:

   None.

2) 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
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) The standard programming language specified by the clause is defined in the American Standard identified by the <language name> keyword. Table 16, "Standard programming languages", specifies the relationship.

Table 16—Standard programming languages

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

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <language clause> shall not specify MUMPS.

10.3 <privileges>

Function

Specify privileges.

Format

```
<privileges> ::=
        ALL PRIVILEGES
      | <action list>

<action list> ::= <action> [ { <comma> <action> }... ]


<action> ::=
        SELECT
      | 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

<privilege column list> ::= <column name list>

<grantee> ::=
        PUBLIC
      | <authorization identifier>
```

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.

2) If *T* is a temporary table, then <privileges> shall specify ALL PRIVILEGES.

3) Each <column name> in a <privilege column list> shall identify a column of *T*.

4) 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>.

5) 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.

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) The set of applicable privileges for an <authorization identifier> includes those privileges defined by privilege descriptors associated with that <authorization identifier>, together with those defined by privilege descriptors associated with PUBLIC.

3) 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.

4) 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.

5) 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) An <action> that specifies INSERT shall not contain a <privilege column list>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

10.4 <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.

Access Rules

1) Let *C* be the <character set name> contained in the <character set specification>. 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 is 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 and form-of-use of *CS* are as defined in Subclause 11.28, "<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 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 Intermediate SQL:

   None.

2) 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>.

10.5 <collate clause>

Function

Specify a collating sequence.

Format

```
<collate clause> ::= COLLATE <collation name>
```

Syntax Rules

None.

Access Rules

1) Let *C* be the <collation name> contained in the <collate clause>. The applicable privileges shall include USAGE on *C*.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

10.6 <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 14.3, "<commit statement>").

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no explicit <constraint attributes>.

      Note: This means that INITIALLY IMMEDIATE NOT DEFERRABLE is implicit.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Intermediate SQL language shall contain no <constraint name definition>.

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 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 element> ::=
         <domain definition>
       | <table definition>
       | <view definition>
       | <grant statement>
       | <assertion definition>
       | <character set definition>
       | <collation definition>
       | <translation definition>
```

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 <procedure> 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 <procedure>s in the <module> have the <module authorization identifier> and <schema name> specified or implicit for the <module>.

5) If <schema character set specification> is not specified, then a <schema character set specification> containing an implementation-defined <character set specification> is implicit.

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 (base tables, views, constraints, domains, assertions, character sets, translations, collations, privileges) 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   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>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Intermediate 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 <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, 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 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 collation definition contained in *S*. The following <drop collation statement> is effectively executed:

   DROP COLLATION *CD*

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) The identified schema and its description are destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

11.3 <table definition>

Function

Define a persistent base table, a created local temporary table, or a global temporary table.

Format

```
<table definition> ::=
       CREATE [ { GLOBAL | LOCAL } TEMPORARY ] TABLE <table name>
         <table element list>
         [ ON COMMIT { DELETE | PRESERVE } ROWS ]

<table element list> ::=
         <left paren> <table element> [ { <comma> <table element> }... ] <right paren>

<table element> ::=
         <column definition>
       | <table constraint definition>
```

Syntax Rules

1) If a <table definition> is contained in a <schema definition>, and if 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>.

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 ON COMMIT is specified, then TEMPORARY shall be specified.

4) If TEMPORARY is specified and ON COMMIT is not specified, then ON COMMIT DELETE ROWS is implicit.

5) A <table definition> shall contain at least one <column definition>.

6) The scope of the <table name> is the <table definition>.

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>.

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 the table being created is initially set to 0; the General Rules of Subclause 11.4, "<column definition>" specify the degree of the table being created during the definition of columns in that table.

3) A table descriptor is created that describes the table being defined.

   a) The name included in the table descriptor is <table name>.

   b) The table descriptor includes the degree of the table, which is the number of <table element>s in the <table definition> that are <column definition>s.

4) 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 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".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not specify TEMPORARY and shall not reference any global or local temporary table.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.4 <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> ]

<column constraint definition> ::=
       [ <constraint name definition> ]
       <column constraint>
         [ <constraint attributes> ]

<column constraint> ::=
         NOT NULL
       | <unique specification>
       | <references specification>
       | <check constraint definition>
```

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 column of *T*.

2) The *i*-th column of the table is described by the *i*-th <column definition> in the <table definition>. The name and the data type or domain of the column are specified by the <column name> and <data type> or <domain name>, respectively.

3) Let *C* be the <column name> of a <column definition>.

4) If <domain name> is specified, then let *D* be the domain identified by the <domain name>.

5) 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*.

6) 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.

7) If a <data type> is specified, then:

a) Let *DT* be the <data type>.

b) The data type of the column is *DT*.

c) 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 <table name> of the containing <table definition> or <alter table statement> is implicit.

d) 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>.

8) If <collate clause> is specified, then data type shall be a character string type.

9) 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 11.7, "<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 11.8, "<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*.

Access Rules

1) If <domain name> is specified, then the applicable privileges shall include USAGE on *D*..

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 altered by the containing <alter table statement> is increased by 1.

5) A column descriptor is created that describes the column being defined. The name included in the column descriptor is <column name>. If the <column definition> specifies <data type>, then the column descriptor includes the data type descriptor of the column; otherwise, the column descriptor includes the name of the domain of the column. The ordinal position included in the column descriptor is equal to the degree of *T*. If the <column definition> contains a <collate clause>, then the <collation name> of the <collate clause> is included in the column descriptor. The column descriptor includes the nullability characteristic of the column, determined according to the rules in Subclause 4.8, "Columns". The column descriptor is included in the table descriptor for *T*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

2) 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 Intermediate SQL language shall contain no <constraint name definition>.

11.5 <default clause>

Function

Specify the default for a column or domain.

Format

```
<default clause> ::=
        DEFAULT <default option>

<default option> ::=
        <literal>
      | <datetime value function>
      | USER
      | CURRENT_USER
      | SESSION_USER
      | SYSTEM_USER
      | NULL
```

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>, <alter column definition>, or <alter domain definition>.

2) If USER is specified, then CURRENT_USER is implicit.

3) 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 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.

iv) If the subject data type is approximate numeric, then the <literal> shall be a <signed numeric literal>.

v) 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.

vi) 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.

b) If CURRENT_USER, SESSION_USER, or SYSTEM_USER 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>.

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, is as follows:

Case:

a) If the <default clause> contains NULL, then the null value.

b) If the <default clause> 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 and a completion condition is raised: *warning—implicit zero-bit padding.*

vi) If the subject data type is datetime or interval, then the value of the <literal>.

c) If the <default clause> specifies CURRENT_USER, SESSION_USER, or SYSTEM_USER, then

Case:

i) If the subject data type is character string with variable length, then the value specified by CURRENT_USER, SESSION_USER, or SYSTEM_USER.

ii) If the subject data type is character string with fixed length, then the value specified by CURRENT_USER, SESSION_USER, or SYSTEM_USER, extended as necessary on the right with <space>s to the length in characters of the subject data type.

d) If the <default clause> contains a <datetime value function>, then the value of an implicit reference to the <datetime value function>.

2) The default value of a column is

Case:

a) If the column descriptor of a column includes a default value derived from a <default option>, then the value of that <default option>.

b) If the column descriptor includes a domain name that identifies a domain descriptor that includes a default value derived from a <default option>, then the value of that <default option>.

c) Otherwise, the null value.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

None.

2) 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.

11.6 <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 Intermediate SQL:

None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Intermediate SQL language shall contain no <constraint name definition>.

11.7 <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 specification> ::=
       UNIQUE | PRIMARY KEY

<unique column list> ::= <column name list>
```

Syntax Rules

1) Let *T* be the table identified by the containing <table definition> or <alter table statement>. Let *TN* be the <table name> of *T*.

2) Let *UCL* be the <unique column list> of the <unique constraint definition>.

3) 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) Each <column name> in the <unique column list> shall identify a column of *T*, and the same column shall not be identified more than once.

5) A <table definition> shall specify at most one implicit or explicit <unique constraint definition> that specifies PRIMARY KEY.

6) 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.

7) 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*.

Access Rules

None.

General Rules

1) A <unique constraint definition> defines a unique constraint.

   Note: Subclause 10.6, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) The unique constraint is not satisfied if and only if

   EXISTS (SELECT * FROM *TN* WHERE NOT (*SC*))

   is true.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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.

11.8 <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 <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
       | SET DEFAULT
       | NO ACTION
```

Syntax Rules

1) 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.

2) 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>.

3) 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.

4) 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.

5) If the referenced table is a temporary table with ON COMMIT DELETE ROWS specified, then the referencing table shall specify ON COMMIT DELETE ROWS.

6) Each referencing column shall identify a column of the referencing table, and the same column shall not be identified more than once.

7) 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 the same as the data type of the corresponding referenced column.

8) If a <referential constraint definition> does not specify any <update rule>, then an <update rule> with a <referential action> of NO ACTION is implicit.

9) If a <referential constraint definition> does not specify any <delete rule>, then a <delete rule> with a <referential action> of NO ACTION is implicit.

Access Rules

1) The applicable privileges shall include REFERENCES for each referenced column.

General Rules

1) A <referential constraint definition> defines a referential constraint.

   Note: Subclause 10.6, "<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) 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.

   b) 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.

   c) 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) Case:

   a) 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.

   b) 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.

4) For every row of the referenced table, its matching rows, unique matching rows, and non-unique matching rows are determined immediately before the execution of any SQL-statement. No new matching rows are added during the execution of that SQL-statement. 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.

5) 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 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 11.5, "<default clause>".

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 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 11.5, "<default clause>".

Note: Otherwise, the <referential action> is not performed.

6) 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 null value.

2) If <match type> specifies FULL, then in all matching rows each referencing column is set to the 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 11.5, "<default clause>".

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 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 11.5, "<default clause>".

Note: Otherwise, the <referential action> is not performed.

7) 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*.

8) 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*.

9) All rows that are marked for deletion are effectively deleted at the end of the SQL-statement, prior to the checking of any integrity constraints.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

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.

2) 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>.

11.9 <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> or a <dynamic parameter 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, or SYSTEM_USER.

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 10.6, "<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 11.6, "<table constraint definition>", or Subclause 11.21, "<domain definition>", depending on whether it forms part of a table constraint or a domain constraint.

2) 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 Intermediate SQL:

   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.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.10 <alter table statement>

Function

Change the definition of a table.

Format

```
<alter table statement> ::=
       ALTER TABLE <table name> <alter table action>

<alter table action> ::=
         <add column definition>
       | <alter column definition>
       | <drop column definition>
       | <add table constraint definition>
       | <drop table constraint definition>
```

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*.

3) The scope of the <table name> is the entire <alter table statement>.

4) *T* shall be a base table.

5) *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>.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

11.11 <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) 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 11.5, "<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 reference>s in these clauses are replaced by explicit <column reference>s when the clause is originally evaluated. See the Syntax Rules of Subclause 7.10, "<query expression>".

3) 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>.

4) 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 11.4, "<column definition>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

11.12 <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>
```

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*.

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 Intermediate SQL:

   None.

2) 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>.

11.13 <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 Intermediate SQL:

   None.

2) 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>.

11.14 <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 Intermediate SQL:

   None.

2) 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>.

11.15 <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>.

   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

None.

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 INSERT(*CN*), UPDATE(*CN*), REFERENCES(*CN*) ON TABLE *TN* FROM *A* CASCADE

2) Let *VN* be the name of any view that contains a reference to column *C* of table *T*. The following <drop view statement> is effectively executed with a current <authorization identifier> of "_SYSTEM" and without further Access Rule checking:

   DROP VIEW *VN* CASCADE

3) If the column is not based on a domain, then its data type descriptor is destroyed.

4) The data associated with *C* is destroyed and the descriptor of *C* is removed from the descriptor of *T*.

5) The identified column and its descriptor are destroyed.

6) 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 Intermediate SQL:

   None.

2) 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>.

11.16 <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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

11.17 <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>. The schema identified by the explicit or implicit schema name of the <table name> shall include the descriptor of *T*.

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*.

4) 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) The identified table constraint and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

11.18 <drop table statement>

Function

Destroy a table.

Format

```
<drop table statement> ::=
       DROP TABLE <table name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <table name> and let *TN* be that <table name>. The schema identified by the explicit or implicit schema name of *TN* shall include the descriptor of *T*.

2) *T* shall be a base table.

3) *T* shall not be a declared local temporary table.

4) If RESTRICT is specified, then *T* shall not be referenced in the <query expression> of any view descriptor or the <search condition> of any constraint descriptor.

   Note: If CASCADE is specified, then such referencing 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 *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

2) The identified base table and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

11.19 <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> or a <dynamic parameter 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) The viewed table defined by <view definition> shall not be identified by any <table reference> generally contained in the <query expression>.

5) Any <table name> that is specified in the <query expression> shall be different from the <table name> of any <temporary table declaration>.

6) If the <query expression> is updatable, then the viewed table is an updatable table. Otherwise, it is a read-only table.

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 6.4, "<column reference>".

12) If WITH CHECK OPTION is specified, then the viewed table shall be updatable.

13) If WITH CHECK OPTION is specified with no <levels clause>, then a <levels clause> of CASCADED is implicit.

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>.

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) A set of privilege descriptors is created that defines the privilege SELECT on this table to *A* and SELECT for each column of *V* to *A*. This privilege is grantable if and only if the applicable SELECT privileges on all <table name>s contained in the <query expression> are grantable. The grantor of this privilege descriptor is set to the special grantor value "_SYSTEM".

5) If *V* is updatable, then let *T* be the leaf underlying table of the <query expression>.

6) 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".

7) If *V* is updatable, then:

   a) A set of privilege descriptors is created that defines the privileges INSERT, UPDATE, and DELETE on *V* that are applicable privileges on *T* to *A*. A privilege on *V* is grantable if and only if the corresponding privilege on *T* is grantable.

   b) For every column in *V*:

      i) There is a corresponding column in *T* from which the column of *V* is derived. Let *CV* and *CT* be the <column name>s of the corresponding columns of *V* and *T* respectively.

      ii) A set of privilege descriptors is created that defines the privileges INSERT(*CV*) and UPDATE(*CV*) on *V*, where the privileges INSERT(*CT*) and UPDATE(*CT*) on *T* are the applicable privileges to *A*, respectively. A privilege on *V* is grantable if and only if the corresponding privilege on *T* is grantable.

   The grantor of these privilege descriptors is set to the special grantor value "_SYSTEM".

8) If *V* is updatable, then let *TLEAF* be the leaf generally underlying table of *V*. For every row in *V* there is a corresponding row in *TLEAF* from which the row of *V* is derived and for each column in *V* there is a corresponding column in *TLEAF* from which the column of *V* is derived. The insertion of a row into *V* is an insertion of a corresponding row into *TLEAF*. The deletion of a row from *V* is a deletion of the corresponding row in *TLEAF*. The updating of a column of a row in *V* is an updating of the corresponding column of the corresponding row in *TLEAF*.

9) Let *V1* be a view. *V1 spans V1*. *V1 spans* a view *V2* if *V2* is a generally underlying table of *V1*.

10) An *update operation* is an <insert statement>, <update statement: positioned>, <update statement: searched>, <dynamic update statement: positioned>, or <preparable dynamic update statement: positioned>. An update operation on a view *V* is an update operation whose <table name> identifies *V*.

11) If a view *V1* spans a view *VA* described by a view descriptor that includes WITH CHECK OPTION and an update operation on *V1* would result in a row that would not appear in the result of *VA*, then

   a) If the view descriptor of *VA* includes CASCADED, then an exception condition is raised: *with check option violation*.

   b) If the view descriptor of *VA* includes LOCAL and the update operation would result in a row that would appear in the simply underlying table of the simply underlying table of the <query expression> contained in *VA*, then an exception condition is raised: *with check option violation*.

12) Validation of a WITH CHECK OPTION constraint is effectively performed at the end of each update operation.

13) 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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <levels clause>, but the effect shall be that defined for a <levels clause> of CASCADED.

2) 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>.

11.20 <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.

   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 Intermediate SQL:

   None.

2) 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>.

11.21 <domain definition>

Function

Define a domain.

Format

```
<domain definition> ::=
       CREATE DOMAIN <domain name> [ AS ] <data type>
         [ <default clause> ]
         [ <domain constraint>... ]
         [ <collate 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>. 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>.

2) If <data type> specifies CHARACTER or CHARACTER VARYING 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.

3) 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.

4) 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.

5) If <collate clause> is specified, then <data type> shall be a character string type.

6) for every <domain constraint> is 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>.

General Rules

1) A <domain definition> defines a domain.

   Note: Subclause 10.6, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

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, and a grantable USAGE privilege for the <collation name> contained in the <collate clause> 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <collate clause>.

2) 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>.

11.22 <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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <alter domain statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.23 <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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <set domain default clause>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.24 <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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <drop domain default clause>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.25 <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 containing <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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <add domain constraint definition>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.26 <drop domain constraint definition>

Function

Destroy a constraint on a domain.

Format

```
<drop domain constraint definition> ::=
       DROP CONSTRAINT <constraint name>
```

Syntax Rules

1) Let *D* be the domain identified by the <domain name> in the containing <alter domain statement>.

2) Let *DC* be the descriptor of the constraint identified by <constraint name>.

3) *DC* shall be included in the domain descriptor of *D*.

Access Rules

None.

General Rules

1) The constraint descriptor of *DC* is removed from the domain descriptor of *D*.

2) The constraint *DC* and its descriptor are destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <drop domain constraint definition>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.27 <drop domain statement>

Function

Destroy a domain.

Format

```
<drop domain statement> ::=
       DROP DOMAIN <domain name> <drop behavior>
```

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.

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 column 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) For every domain constraint descriptor included in the domain descriptor of *D*:

      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) 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 DOMAIN *DN* FROM *A* CASCADE

3) The identified domain is destroyed by destroying its descriptor and its data type descriptor.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

None.

2) 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>.

11.28 <character set definition>

Function

Define a character set.

Format

```
<character set definition> ::=
       CREATE CHARACTER SET <character set name> [ AS ]
         <character set source>
         [ <collate clause> | <limited collation definition> ]

<character set source> ::=
         GET <existing character set name>

<existing character set name> ::=
         <standard character repertoire name>
       | <implementation-defined character repertoire name>
       | <schema character set name>

<schema character set name> ::= <character set name>

<limited collation definition> ::=
       COLLATION FROM <collation source>
```

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) A <schema character set name> shall identify some character set descriptor.

4) If neither <collate clause> nor <limited collation definition> is specified, then the following <limited collation definition> is implicit:

   COLLATION FROM DEFAULT

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 has the same character repertoire as the character set identified by the <existing character set name>.

4) 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 Intermediate SQL:

   a) In conforming Intermediate SQL language, <collation source> shall specify DEFAULT.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not specify a <character set definition>.

11.29 <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, or be included in any collation descriptor or translation 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 Intermediate SQL:

   None.

2) 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>.

11.30 <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>

<collating sequence definition> ::=
         <external collation>
       | <schema collation name>
       | DESC <left paren> <collation name> <right paren>
       | DEFAULT

<translation collation> ::=
       TRANSLATION <translation name>
           [ THEN COLLATION <collation name> ]

<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>
```

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 a <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) 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.

8) 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.

9) 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*.

10) 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*.

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) Let *C* be a collation identified by any <collation name> contained in <collation source>. 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) 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".

5) 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.

6) 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*.

7) 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.

8) A collation descriptor is created for the defined collation.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <collation definition>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.31 <drop collation statement>

Function

Destroy a collating sequence.

Format

```
<drop collation statement> ::=
       DROP COLLATION <collation name>
```

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*.

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) 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

2) Let *CD* be any collation descriptor that includes *CN*. *CD* is modified by deleting any occurrences of "THEN COLLATION *CN*" or "DESC (*CN*)"

3) Let *CSD* be any character set descriptor that includes *CN*. *CSD* is modified by deleting any occurrences of "COLLATION FROM *CN*" or "DESC (*CN*)".

4) Let *DD* be any column descriptor or domain descriptor that includes *CN*. *DD* is modified by deleting any occurrences of "COLLATE *CN*".

5) Let *VD* be any view descriptor whose <query expression> includes "COLLATE *CN*" or any constraint descriptor whose <search condition> includes "COLLATE *CN*". *VD* is modified by deleting any occurrences of "COLLATE *CN*".

6) The descriptor of *C* is destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <drop collation statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

11.32 <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 specification> ::=
         <external translation>
       | IDENTITY
       | <schema translation name>

<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>
```

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 a <implementation-defined translation name> shall have associated with it a privilege descriptor that was effectively defined by the <grant statement>

   GRANT USAGE ON TRANSLATION *TRANS* TO PUBLIC

   where *TRANS* is the <standard translation name> or <implementation-defined translation name>.

5) A <schema translation name> shall identify a translation descriptor.

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) A translation descriptor is created for the defined translation.

4) 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".

5) *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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <translation definition>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.33 <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 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 *CD* be any collation descriptor that includes a <translation collation> TRANSLATION *TN*. *CD* is modified by deleting that <translation collation>.

2) Let *CSD* be any <character set definition> that references *T*. *CSD* is modified by deleting any occurrences of a <translation collation> that contains *TN*.

3) 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

4) The descriptor of *T* is destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <drop translation statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.34 <assertion definition>

Function

Specify an integrity constraint by means of an assertion and specify the initial default time for checking the assertion.

Format

```
<assertion definition> ::=
       CREATE ASSERTION <constraint name> <assertion check> [ <constraint attributes> ]

<assertion check> ::=
       CHECK <left paren> <search condition> <right paren>
```

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 <target specification> or a <dynamic parameter specification>.

5) No <query expression> in the <search condition> shall reference 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.

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 *TN* be any <table name> referenced in the <search condition> of the <assertion definition>. If *TN* identifies a table described by a base table descriptor or a view descriptor, then

   Case:

   a) If a <column name> is contained in the <search condition>, then the applicable privileges shall include REFERENCES for each <column name> *CN* of the table identified by *TN*, where *CN* is 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) An <assertion definition> defines an assertion.

   Note: Subclause 10.6, "<constraint name definition> and <constraint attributes>", specifies when a constraint is effectively checked.

2) The assertion is not satisfied if and only if the result of evaluating the <search condition> is false.

3) 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 the <search condition> of the <assertion definition>.

4) 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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.35 <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) The descriptor of *A* is destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <drop assertion statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

11.36 <grant statement>

Function

Define privileges.

Format

```
<grant statement> ::=
       GRANT <privileges> ON <object name>
         TO <grantee> [ { <comma> <grantee> }... ]
           [ WITH GRANT OPTION ]

<object name> ::=
         [ TABLE ] <table name>
       | DOMAIN <domain name>
       | COLLATION <collation name>
       | CHARACTER SET <character set name>
       | TRANSLATION <translation name>
```

Syntax Rules

1) If <object name> specifies a <domain name>, <collation name>, <character set name>, or <translation name>, then <privileges> shall specify USAGE; otherwise, USAGE shall not be specified.

2) Let *O* be the object identified by the <object name>.

3) Let *A* be the current <authorization identifier>. For each <grantee> specified, 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.

Access Rules

1) The applicable privileges shall include a privilege identifying *O*.

General Rules

1) The <privileges> specify one or more privileges on the object identified by the <object name>.

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*.

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 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 view *V1* owned by *G*, if *G* has been granted SELECT privilege WITH GRANT OPTION on all tables identified by a <table name> contained in the <query expression> of *V1*, then for every privilege descriptor with a <privileges> *P* that contains SELECT, a <grantor> of "_SYSTEM", <object> of *V1*, 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 *P* ON *V1* TO *G* WITH GRANT OPTION

8) For every <grantee> *G* and for every domain *D1* owned by *G*, if *G* has been granted 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, and a grantable USAGE privilege for the <collation name> contained in the <collate clause> 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

9) For every <grantee> *G* and for every collation *C1* owned by *G*, if the USAGE privilege of *G* for the character set identified by a <character set specification> contained in the <collation definition> of *C1* is grantable, then for every privilege descriptor with a <privileges> *P*, 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 *P* ON COLLATION *C1* TO *G* WITH GRANT OPTION

10) For every <grantee> *G* and for every translation *T1* owned by *G*, if the USAGE privilege of *G* for every character set identified by a <character set specification> contained in the <translation definition> of *T1* is grantable, 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

11) If <table name> is specified, then for each view *V* owned by some <grantee> *G* such that *T* or some column *CT* of *T*, 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>, *G* will have REFERENCES(CRT_{ij}) for all *i* and for all *j*, and *A* has REFERENCES on some column of RT_i for all *i*, the 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*

12) 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.

13) Redundant duplicate privilege descriptors are removed from the multiset of all privilege descriptors.

14) 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*.

15) 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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

    a) In Conforming Intermediate SQL language, an <object name> shall not specify COLLATION or TRANSLATION.

2) 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.

11.37 <revoke statement>

Function

Destroy privileges.

Format

```
<revoke statement> ::=
       REVOKE [ GRANT OPTION FOR ] <privileges>
           ON <object name>
         FROM <grantee> [ { <comma> <grantee> }... ] <drop behavior>
```

Syntax Rules

1) If <object name> specifies a <domain name>, <collation name>, <character set name>, or <translation name>, then <privileges> shall specify USAGE; otherwise, USAGE shall not be specified.

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 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 *allowed to be created* by a grant permitted by *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) *P* is a table privilege descriptor and *D* is a column privilege descriptor. The identified table of *P* is the same as the identified table of *D* and the action of *P* is the same as the action of *D* and the action of *P* is SELECT.

3) 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 a <view definition> *V* and either:

A) The action of *P* is SELECT and the identified table of *P* is contained in the <query expression> of *V*, or

B) *V* is an updatable view and the identified table of *P* is the underlying table of the <query expression>.

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 *V* is an 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*.

3) *P* is a table privilege descriptor and the column privilege descriptor *D* identifies a <column name> *CV* explicitly or implicitly contained in the <view column list> of a <view definition> *V*. Let *TN* be a <table name> contained in the <query expression> of the view. The action for *P* is SELECT and the identified table of *P* is *TN*.

4) The privilege descriptor *D* identifies the <collation name> of a <collation definition> *CO* and the identified character set name of *P* is contained in the <character set specification> immediately contained in *CO*.

5) 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*.

8) A privilege descriptor *D* is said to be *directly dependent on* another privilege descriptor *P* if *D* represents a privilege allowed to be created by a grant permitted by *P*.

9) 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.

10) A privilege descriptor *P* is said to be *modified* if either *P* is a SELECT column privilege descriptor and a SELECT table privilege descriptor with the same grantee, grantor, catalog name, schema name, and table name is a modified privilege descriptor, or:

    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) 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 exists 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, 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*.

11) A privilege descriptor *P* is *abandoned* if:

    a) It is not an independent node, and

    Case:

        i) GRANT OPTION FOR is not specified, *P* is not itself 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.

        ii) GRANT OPTION FOR is specified, *P* is not itself a modified privilege descriptor, and there exists no path to *P* from any independent node other than paths that include a modified privilege descriptor.

    b) *P* is a SELECT column privilege descriptor and there exists a SELECT table privilege descriptor *X* with the same grantee, grantor, catalog name, schema name, and table name and

    Case:

        i) GRANT OPTION FOR is not specified and *X* is an identified privilege descriptor or an abandoned privilege descriptor.

        ii) GRANT OPTION FOR is specified and *X* is an abandoned privilege descriptor.

12) Let *S1* be the name of any schema and let *A1* be the <authorization identifier> that owns the schema identified by *S1*.

13) 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 SELECT privilege on one or more tables or USAGE privilege on one or more domains, collations, character sets, or translations whose names are contained in the <query expression> of *V*.

14) 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 REFERENCES privilege on one or more referenced columns of *TC* or USAGE privilege on one or more domains, collations, character sets, or translations whose names are contained in any <search condition> of *TC*.

15) 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 REFERENCES privilege on one or more referenced columns of *AX* or USAGE privilege on one or more domains, collations, character sets, or translations whose names are contained in any <search condition> of *AX*.

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 REFERENCES privilege on one or more referenced columns of *DC* or USAGE privilege on one or more domains, collations, character sets, or translations whose names are contained in any <search condition> of *DC*.

17) Let *DO* be any domain descriptor included in *S1*. *DO* 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 USAGE privilege on the collation whose name is contained in the <collate clause> of *DO*, if any.

18) If RESTRICT is specified, then there shall be no abandoned privilege descriptors, abandoned views, abandoned table constraints, abandoned assertions, abandoned domain constraints, or abandoned domains.

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, and

   b) The identified privilege descriptors are destroyed, and

   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 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*

7) For every abandoned 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

8) 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*.

9) 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 Intermediate SQL:

None.

2) 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>.

12 Module

12.1 <module>

Function

Define a module.

Format

```
<module> ::=
       <module name clause>
       <language clause>
       <module authorization clause>
       [ <temporary table declaration>... ]
       <module contents>...

<module authorization clause> ::=
         SCHEMA <schema name>
       | AUTHORIZATION <module authorization identifier>
       | SCHEMA <schema name> AUTHORIZATION <module authorization identifier>

<module authorization identifier> ::=
       <authorization identifier>

<module contents> ::=
         <declare cursor>
       | <dynamic declare cursor>
       | <procedure>
```

Syntax Rules

1) If SCHEMA <schema name> is not specified, then a <schema name> equal to <module authorization identifier> is implicit.

2) If the explicit or implicit <schema name> does not specify a <catalog name>, then an implementation-defined <catalog name> is implicit.

3) The implicit or explicit <catalog name> is the implicit <catalog name> for all unqualified <schema name>s in the <module>.

4) A <declare cursor> or <dynamic declare cursor> shall precede in the text of the <module> any <procedure> that references the <cursor name> of the <declare cursor> or <dynamic declare cursor>.

5) For every <declare cursor> in a <module>, there shall be exactly one <procedure> in that <module> that contains an <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 that performs a call of a <procedure> in a <module> is not a program that conforms to the programming language standard specified by the <language clause> of that <module>, then the effect is implementation-dependent.

2) If the SQL-agent performs calls of <procedure>s from more than one Ada task, then the results are implementation-dependent.

3) Case:

   a) If a <module authorization identifier> is specified, then it is the current <authorization identifier> for privilege determination for the execution of each <procedure> in the <module>.

   b) Otherwise, the current <authorization identifier> for privilege determination for the execution of each <procedure> in the <module> is the SQL-session <authorization identifier>.

4) After the last time that an SQL-agent performs a call of a <procedure>:

   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. The determination of whether an SQL-agent has terminated unexpectedly is implementation-dependent.

   b) Let *D* be the <descriptor name> of any system descriptor area that is currently allocated within an SQL-session associated with the SQL-agent. A <deallocate descriptor statement> that specifies

      DEALLOCATE DESCRIPTOR *D*

      is effectively executed.

   c) All SQL-sessions associated with the SQL-agent are terminated.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <module> shall not contain a <temporary table declaration>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <module> shall be associated with an SQL-agent during its execution. An SQL-agent shall be associated with at most one <module>.

   b) A <module contents> shall not be a <dynamic declare cursor>.

c) A <module authorization clause> shall specify AUTHORIZATION and shall not specify SCHEMA.

12.2 <module name clause>

Function

Name a <module>.

Format

```
<module name clause> ::=
       MODULE [ <module name> ]
         [ <module character set specification> ]

<module character set specification> ::=
       NAMES ARE <character set specification>
```

Syntax Rules

1) If a <module name clause> does not specify a <module name>, then the <module> is unnamed.

2) The <module name> shall be different from the <module name> of any other <module> in the same SQL-environment.

   Note: An SQL-environment may have multiple <module>s that are unnamed.

3) 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.

4) 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.

Access Rules

None.

General Rules

1) If a <module name> is specified, then in the SQL-environment the containing <module> has the name given by <module name>.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <module character set specification> shall not be specified.

12.3 <procedure>

Function

Define a procedure.

Format

```
<procedure> ::=
       PROCEDURE <procedure name> <parameter declaration list> <semicolon>
         <SQL procedure statement> <semicolon>

<parameter declaration list> ::=
         <left paren> <parameter declaration>
             [ { <comma> <parameter declaration> }... ] <right paren>
       | <parameter declaration>...

<parameter declaration> ::=
         <parameter name> <data type>
       | <status parameter>

<status parameter> ::=
       SQLCODE | SQLSTATE
```

Syntax Rules

1) The <procedure name> shall be different from the <procedure name> of any other <procedure> in the containing <module>.

   Note: The <procedure name> should be a standard-conforming procedure, function, or routine name of the language specified by the subject <language clause>. Failure to observe this recommendation will have implementation-dependent effects.

2) The <parameter name> of each <parameter declaration> in a <procedure> shall be different from the <parameter name> of any other <parameter declaration> in that <procedure>.

3) Any <parameter name> contained in the <SQL procedure statement> of a <procedure> shall be specified in a <parameter declaration> in that <procedure>.

   Note: <parameter declaration>s in a <procedure> without enclosing parentheses and without commas separating multiple <parameter declaration>s is a deprecated feature that is supported for compatibility with earlier versions of this American Standard. See Annex D, "Deprecated features".

4) A call of a <procedure> shall supply *n* parameters, where *n* is the number of <parameter declaration>s in the <procedure>.

5) A <procedure> shall contain at least one <status parameter>, at most one <status parameter> that specifies SQLCODE, and at most one <status parameter> that specifies SQLSTATE. A parameter that corresponds with SQLCODE is referred to as an SQLCODE parameter. A parameter that corresponds with SQLSTATE is referred to as an SQLSTATE parameter. The SQLCODE and SQLSTATE parameters are referred to as status parameters.

   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 American Standard. See Annex D, "Deprecated features".

6) Whether a <parameter declaration> is for an *input parameter*, an *output parameter*, or both is determined as follows:

   Case:

   a) A <status parameter> is an *output parameter*.

   b) For every <parameter declaration> that is not a <status parameter>,

      Case:

      i) If the <parameter name> of a parameter is contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement>, but it is not contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, then the parameter is an *input parameter*.

      ii) If the <parameter name> of a parameter is contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, but it is not contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement>, then the parameter is an *output parameter*.

      iii) If the <parameter name> of a parameter is contained in a <value specification> or a <simple value specification> that is contained in <SQL procedure statement> and it is contained in a <target specification> or a <simple target specification> that is contained in <SQL procedure statement>, then the parameter is both an *input parameter* and an *output parameter*.

      iv) Otherwise, the parameter is neither an *input parameter* nor an *output parameter*.

7) The Syntax Rules of Subclause 12.4, "Calls to a <procedure>", shall be true.

Access Rules

None.

General Rules

1) A <procedure> defines a procedure that may be called by an SQL-agent.

2) If the <module> that contains the <procedure> is associated with an SQL-agent that is associated with another <module> that contains a <procedure> with the same <procedure name>, then the effect is implementation-defined.

3) If the <module> that contains the <procedure> 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 <procedure>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 <procedure> in a <module> with explicit <module authorization identifier> *MAI*, then an exception condition is raised: *invalid authorization specification*.

4) 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 <procedure> 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*.

5) Let *S* be the <SQL procedure statement> of the <procedure>.

6) When the <procedure> is called by an SQL-agent:

   Case:

   a) 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 <procedure> by the SQL-agent are associated with that SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

   b) If *S* is an <SQL diagnostics statement>, then:

      i) The <module> that contains *S* is associated with the SQL-agent.

      ii) *S* is executed.

   c) 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 <procedure> or invocations of <direct SQL statement>s 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 <procedure> or invocations of <direct SQL statement>s by that SQL-agent until the SQL-agent terminates that SQL-transaction.

2) 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) 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*.

vi) The diagnostics area is emptied.

vii) The values of all input parameters to the <procedure> are established.

viii) *S* is executed.

7) If the non-dynamic or dynamic execution of an <SQL data statement> or the execution of an <SQL dynamic data statement>, <dynamic select statement>, or <dynamic single row select statement> occurs within the same SQL-transaction as the non-dynamic or dynamic execution of an SQL-schema statement and this is not allowed by the SQL-implementation, then an exception condition is raised: *invalid transaction state*.

8) When a <procedure> 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 <data type> and the <parameter name> specified in PD_i, respectively. Let PI_i be the *i*-th parameter in the procedure call.

9) If *S* is a <select statement: single row> or a <fetch statement> and a completion condition *no data* is raised or an exception condition is raised, then the value of each PI_i for which PN_i is referenced in a <target specification> in *S* is implementation-dependent.

10) The General Rules of Subclause 12.4, "Calls to a <procedure>", are evaluated.

11) 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*, as determined by the General Rules in this and other Subclauses of this American Standard.

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) The status parameter(s) is (are) set to the value(s) specified for the condition in clause Clause 22, "Status codes". If there is more than one status parameters, then the order in which values are assigned to these status parameters is implementation-dependent.

12) Case:

a) If *S* is not an <SQL diagnostics statement>, then diagnostics information resulting from the execution of *S* is placed into the diagnostics area as specified in Clause 18, "Diagnostics management".

b) If *S* is an <SQL diagnostics statement>, then the diagnostics area is not updated.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

a) A <parameter declaration> shall not specify a <data type> that is BIT or BIT VARYING.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) A <parameter declaration> shall not specify a <data type> that is CHARACTER VARYING.

12.4 Calls to a <procedure>

Function

Define the call to a <procedure> by an SQL-agent.

Syntax Rules

1) If the subject <language clause> specifies ADA, then:

   a) The <procedure>s of the <module> are identified by the <procedure name> as if they were declared immediately within an Ada library unit package specification that has a name identical to the <module name> of the containing <module>. The SQL-implementation shall generate the source code of the Ada library unit package specification.

   b) Any <data type> in a <parameter declaration> shall specify CHARACTER, BIT, SMALLINT, INTEGER, REAL, or DOUBLE PRECISION.

   c) The base type of any parameter shall be an Ada data type declared in an Ada package named SQL_STANDARD of the following form:

```
package SQL_STANDARD is
       package CHARACTER_SET renames csp;
       subtype CHARACTER_TYPE is CHARACTER_SET.cst;
       type CHAR is array (POSITIVE range <>) of CHARACTER_TYPE;
       type BIT is array (NATURAL range <>) of BOOLEAN;
       type SMALLINT is range bs .. ts;
       type INT is range bi .. ti;
       type REAL is digits dr;
       type DOUBLE_PRECISION is digits dd;
       subtype INDICATOR_TYPE is t;
       type SQLCODE_TYPE is range bsc .. tsc;
       subtype SQL_ERROR is SQLCODE_TYPE range SQL_TYPE'FIRST .. -1;
       subtype NOT_FOUND is SQLCODE_TYPE range 100 .. 100;
       type SQLSTATE_TYPE is new CHAR (1 .. 5);
       package SQLSTATE_CODES is
           AMBIGUOUS_CURSOR_NAME_NO_SUBCLASS:
                   constant SQLSTATE_TYPE :="3C000";
           CARDINALITY_VIOLATION_NO_SUBCLASS:
                   constant SQLSTATE_TYPE :="21000";
           CONNECTION_EXCEPTION_NO_SUBCLASS:
                   constant SQLSTATE_TYPE :="08000";
           CONNECTION_EXCEPTION_CONNECTION_DOES_NOT_EXIST:
                   constant SQLSTATE_TYPE :="08003";
           CONNECTION_EXCEPTION_CONNECTION_FAILURE:
                   constant SQLSTATE_TYPE :="08006";
           CONNECTION_EXCEPTION_CONNECTION_NAME_IN_USE:
                   constant SQLSTATE_TYPE :="08002";
           CONNECTION_EXCEPTION_SQLCLIENT_UNABLE_TO_ESTABLISH_SQLCONNECTION:
                   constant SQLSTATE_TYPE :="08001";
           CONNECTION_EXCEPTION_SQLSERVER_REJECTED_ESTABLISHMENT_OF_SQLCONNECTION:
                   constant SQLSTATE_TYPE :="08004";
           CONNECTION_EXCEPTION_TRANSACTION_RESOLUTION_UNKNOWN:
                   constant SQLSTATE_TYPE :="08007";
           DATA_EXCEPTION_NO_SUBCLASS:
                   constant SQLSTATE_TYPE :="22000";
           DATA_EXCEPTION_CHARACTER_NOT_IN_REPERTOIRE:
                   constant SQLSTATE_TYPE :="22021";
           DATA_EXCEPTION_DATETIME_FIELD_OVERFLOW:
                   constant SQLSTATE_TYPE :="22008";
           DATA_EXCEPTION_DIVISION_BY_ZERO:
```

```
        constant SQLSTATE_TYPE :="22012";
DATA_EXCEPTION_ERROR_IN_ASSIGNMENT:
        constant SQLSTATE_TYPE :="22005";
DATA_EXCEPTION_INDICATOR_OVERFLOW:
        constant SQLSTATE_TYPE :="22022";
DATA_EXCEPTION_INTERVAL_FIELD_OVERFLOW:
        constant SQLSTATE_TYPE :="22015";
DATA_EXCEPTION_INVALID_CHARACTER_VALUE_FOR_CAST:
        constant SQLSTATE_TYPE :="22018";
DATA_EXCEPTION_INVALID_DATETIME_FORMAT:
        constant SQLSTATE_TYPE :="22007";
DATA_EXCEPTION_INVALID_ESCAPE_CHARACTER:
        constant SQLSTATE_TYPE :="22019";
DATA_EXCEPTION_INVALID_ESCAPE_SEQUENCE:
        constant SQLSTATE_TYPE :="22025";
DATA_EXCEPTION_INVALID_PARAMETER_VALUE:
        constant SQLSTATE_TYPE :="22023";
DATA_EXCEPTION_INVALID_TIME_ZONE_DISPLACEMENT_VALUE:
        constant SQLSTATE_TYPE :="22009";
DATA_EXCEPTION_NULL_VALUE_NO_INDICATOR_PARAMETER:
        constant SQLSTATE_TYPE :="22002";
DATA_EXCEPTION_NUMERIC_VALUE_OUT_OF_RANGE:
        constant SQLSTATE_TYPE :="22003";
DATA_EXCEPTION_STRING_DATA_LENGTH_MISMATCH:
        constant SQLSTATE_TYPE :="22026";
DATA_EXCEPTION_STRING_DATA_RIGHT_TRUNCATION:
        constant SQLSTATE_TYPE :="22001";
DATA_EXCEPTION_SUBSTRING_ERROR:
        constant SQLSTATE_TYPE :="22011";
DATA_EXCEPTION_TRIM_ERROR:
        constant SQLSTATE_TYPE :="22027";
DATA_EXCEPTION_UNTERMINATED_C_STRING:
        constant SQLSTATE_TYPE :="22024";
DEPENDENT_PRIVILEGE_DESCRIPTORS_STILL_EXIST_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2B000";
DYNAMIC_SQL_ERROR_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="07000";
DYNAMIC_SQL_ERROR_CURSOR_SPECIFICATION_CANNOT_BE_EXECUTED:
        constant SQLSTATE_TYPE :="07003";
DYNAMIC_SQL_ERROR_INVALID_DESCRIPTOR_COUNT:
        constant SQLSTATE_TYPE :="07008";
DYNAMIC_SQL_ERROR_INVALID_DESCRIPTOR_INDEX:
        constant SQLSTATE_TYPE :="07009";
DYNAMIC_SQL_ERROR_PREPARED_STATEMENT_NOT_A_CURSOR_SPECIFICATION:
        constant SQLSTATE_TYPE :="07005";
DYNAMIC_SQL_ERROR_RESTRICTED_DATA_TYPE_ATTRIBUTE_VIOLATION:
        constant SQLSTATE_TYPE :="07006";
DYNAMIC_SQL_ERROR_USING_CLAUSE_DOES_NOT_MATCH_DYNAMIC_PARAMETER_SPEC:
        constant SQLSTATE_TYPE :="07001";
DYNAMIC_SQL_ERROR_USING_CLAUSE_DOES_NOT_MATCH_TARGET_SPEC:
        constant SQLSTATE_TYPE :="07002";
DYNAMIC_SQL_ERROR_USING_CLAUSE_REQUIRED_FOR_DYNAMIC_PARAMETERS:
        constant SQLSTATE_TYPE :="07004";
DYNAMIC_SQL_ERROR_USING_CLAUSE_REQUIRED_FOR_RESULT_FIELDS:
        constant SQLSTATE_TYPE :="07007";
FEATURE_NOT_SUPPORTED_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="0A000";
FEATURE_NOT_SUPPORTED_MULTIPLE_ENVIRONMENT_TRANSACTIONS:
        constant SQLSTATE_TYPE :="0A001";
INTEGRITY_CONSTRAINT_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="23000";
INVALID_AUTHORIZATION_SPECIFICATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="28000";
INVALID_CATALOG_NAME_NO_SUBCLASS:
```

```
        constant SQLSTATE_TYPE :="3D000";
INVALID_CHARACTER_SET_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2C000";
INVALID_CONDITION_NUMBER_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="35000";
INVALID_CONNECTION_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2E000";
INVALID_CURSOR_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="34000";
INVALID_CURSOR_STATE_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="24000";
INVALID_SCHEMA_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="3F000";
INVALID_SQL_DESCRIPTOR_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="33000";
INVALID_SQL_STATEMENT_NAME_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="26000";
INVALID_TRANSACTION_STATE_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="25000";
INVALID_TRANSACTION_TERMINATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2D000";
NO_DATA_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="02000";
REMOTE_DATABASE_ACCESS_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="HZ000";
SUCCESSFUL_COMPLETION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="00000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="42000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_IN_DIRECT_STATEMENT_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="2A000";
SYNTAX_ERROR_OR_ACCESS_RULE_VIOLATION_IN_DYNAMIC_STATEMENT_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="37000";
TRANSACTION_ROLLBACK_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="40000";
TRANSACTION_ROLLBACK_INTEGRITY_CONSTRAINT_VIOLATION:
        constant SQLSTATE_TYPE :="40002";
TRANSACTION_ROLLBACK_SERIALIZATION_FAILURE:
        constant SQLSTATE_TYPE :="40001";
TRANSACTION_ROLLBACK_STATEMENT_COMPLETION_UNKNOWN:
        constant SQLSTATE_TYPE :="40003";
TRIGGERED_DATA_CHANGE_VIOLATION_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="27000";
WARNING_NO_SUBCLASS:
        constant SQLSTATE_TYPE :="01000";
WARNING_CURSOR_OPERATION_CONFLICT:
        constant SQLSTATE_TYPE :="01001";
WARNING_DISCONNECT_ERROR:
        constant SQLSTATE_TYPE :="01002";
WARNING_IMPLICIT_ZERO_BIT_PADDING:
        constant SQLSTATE_TYPE :="01008";
WARNING_INSUFFICIENT_ITEM_DESCRIPTOR_AREAS:
        constant SQLSTATE_TYPE :="01005";
WARNING_NULL_VALUE_ELIMINATED_IN_SET_FUNCTION:
        constant SQLSTATE_TYPE :="01003";
WARNING_PRIVILEGE_NOT_GRANTED:
        constant SQLSTATE_TYPE :="01007";
WARNING_PRIVILEGE_NOT_REVOKED:
        constant SQLSTATE_TYPE :="01006";
WARNING_QUERY_EXPRESSION_TOO_LONG_FOR_INFORMATION_SCHEMA:
        constant SQLSTATE_TYPE :="0100A";
WARNING_SEARCH_CONDITION_TOO_LONG_FOR_INFORMATION_SCHEMA:
        constant SQLSTATE_TYPE :="01009";
WARNING_STRING_DATA_RIGHT_TRUNCATION_WARNING:
```

```
                constant SQLSTATE_TYPE :="01004";
        WITH_CHECK_OPTION_VIOLATION_NO_SUBCLASS:
                constant SQLSTATE_TYPE :="44000";
    end SQLSTATE_CODES;
end SQL_STANDARD;
```

where *csp* is an implementation-defined package and *cst* is an implementation-defined character type such that within the scope of an Ada **use** clause for SQL_STANDARD.CHARACTER_SET, string literals can be of type SQL_STANDARD.CHAR. *bs*, *ts*, *bi*, *ti*, *dr*, *dd*, *bsc*, and *tsc* are implementation-defined integer values. *t* is INT or SMALLINT, corresponding with an implementation-defined <exact numeric type> of indicator parameters.

SQL_STANDARD shall contain no other declarations.

d) The base type of the SQLCODE parameter shall be

SQL_STANDARD.SQLCODE_TYPE

The base type of the SQLSTATE parameter shall be

SQL_STANDARD.SQLSTATE_TYPE

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 American Standard. See Annex D, "Deprecated features".

e) The Ada parameter mode of the SQLCODE parameter is **out**. The Ada parameter mode of the SQLSTATE parameter is **out**.

f) If the *i*-th <parameter declaration> specifies a <data type> *PDT* that is CHARACTER(*L*) for some <length> *L*, then the base type of the *i*-th parameter *P* shall be SQL_STANDARD.CHAR, with P'LENGTH equal to the maximum possible length in octets of *PDT*.

g) If the *i*-th <parameter declaration> specifies a <data type> that is BIT(*L*) for some <length> *L*, then the base type of the *i*-th parameter *P* shall be SQL_STANDARD.BIT with *P*'LENGTH equal to *L*.

Note: BOOLEAN is the predefined enumeration type STANDARD.BOOLEAN with values (FALSE, TRUE). The equivalences

BOOLEAN'POS(FALSE) ≡ 0
BOOLEAN'POS(TRUE) ≡ 1

define the correspondence between the bit values of 0 and 1 and the Boolean values of *false* and *true*, respectively.

h) If the *i*-th <parameter declaration> specifies a <data type> that is SMALLINT, then the base type of the *i*-th parameter shall be SQL_STANDARD.SMALLINT.

i) If the *i*-th <parameter declaration> specifies a <data type> that is INTEGER, then the base type of the *i*-th parameter shall be SQL_STANDARD.INT.

j) If the *i*-th <parameter declaration> specifies a <data type> that is REAL, then the base type of the *i*-th parameter shall be SQL_STANDARD.REAL.

k) If the *i*-th <parameter declaration> specifies a <data type> that is DOUBLE PRECISION, then the base type of the *i*-th parameter shall be SQL_STANDARD. DOUBLE_PRECISION.

l) For every parameter,

Case:

i) If the parameter is an input parameter but not an output parameter, then the Ada parameter mode is **in**.

ii) If the parameter is an output parameter but not an input parameter, then the Ada parameter mode is **out**.

iii) If the parameter is both an input parameter and an output parameter, then the Ada parameter mode is **in out**.

iv) Otherwise, the Ada parameter mode is **in**, **out**, or **in out**.

2) If the subject <language clause> specifies C, then:

a) The type of an SQLCODE parameter shall be C pointer to long.

The type of an SQLSTATE parameter shall be C char with length 6.

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 American Standard. See Annex D, "Deprecated features".

b) Any <data type> in a <parameter declaration> shall specify either CHARACTER, CHARACTER VARYING, BIT, INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

c) If the *i*-th <parameter declaration> specifies a <data type> *PDT* that is CHARACTER(*L*) or CHARACTER VARYING(*L*) for some <length> *L*, then the type of the *i*-th parameter *P* shall be C char with length one greater than the maximum possible length in octets of *PDT*.

d) If the *i*-th <parameter declaration> specifies a <data type> that is BIT(*L*) for some <length> *L*, then the type of the *i*-th parameter is C char with length equal to the smallest integer not less than the quotient of the division of (*L*/(*B*+1)), where *B* is the implementation-defined number of bits in a C character.

e) If the *i*-th <parameter declaration> specifies a <data type> that is INTEGER, then the type of the *i*-th parameter shall be C pointer to long.

f) If the *i*-th <parameter declaration> specifies a <data type> that is SMALLINT, then the type of the *i*-th parameter shall be C pointer to short.

g) If the *i*-th <parameter declaration> specifies a <data type> that is REAL, then the type of the *i*-th parameter shall be C pointer to float.

h) If the *i*-th <parameter declaration> specifies a <data type> that is DOUBLE PRECISION, then the type of the *i*-th parameter shall be C pointer to double.

3) If the subject <language clause> specifies COBOL, then:

a) The type of an SQLCODE parameter shall be COBOL PICTURE S9(PC) USAGE COMPUTATIONAL, where PC is an implementation-defined precision between 4 and 18, inclusive.

The type of an SQLSTATE parameter shall be COBOL PICTURE X(5).

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 American Standard. See Annex D, "Deprecated features".

b) Any <data type> in a <parameter declaration> shall specify CHARACTER, BIT, NUMERIC, INTEGER, or SMALLINT.

c) If the *i*-th <parameter declaration> specifies a <data type> *PDT* that is CHARACTER(*L*) for some <length> *L*, then the type of the *i*-th parameter *P* shall be COBOL alphanumeric with length equal to the maximum possible length in octets of *PDT*.

d) If the *i*-th <parameter declaration> specifies a <data type> that is BIT(*L*) for some <length> *L*, then the type of the *i*-th parameter is COBOL alphanumeric with length equal to the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits in a COBOL character.

e) If the *i*-th <parameter declaration> specifies a <data type> that is NUMERIC(*P*, *S*) for some <precision> and <scale> *P* and *S*, then the type of the *i*-th parameter shall be COBOL USAGE DISPLAY SIGN LEADING SEPARATE with the following PICTURE:

Case:

i) If *S*=*P*, then a PICTURE with an "S" followed by a "V" followed by *P* "9"s.

ii) If *P*>*S*>0, then a PICTURE with an "S" followed by *P*–*S* "9"s followed by a "V" followed by *S* "9"s.

iii) If *S*=0, then a PICTURE with an "S" followed by *P* "9"s optionally followed by a "V".

f) If the *i*-th <parameter declaration> specifies a <data type> that is INTEGER, then the type of the *i*-th parameter shall be COBOL PICTURE S9(*PI*) USAGE BINARY, where *PI* is an implementation-defined precision.

g) If the *i*-th <parameter declaration> specifies a <data type> that is SMALLINT, then the type of the *i*-th parameter shall be COBOL PICTURE S9(*SPI*) USAGE BINARY, where *SPI* is an implementation-defined precision.

4) If the subject <language clause> specifies FORTRAN, then:

a) The type of an SQLCODE parameter shall be Fortran INTEGER.

The type of an SQLSTATE parameter shall be Fortran CHARACTER with length 5.

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 American Standard. See Annex D, "Deprecated features".

b) Any <data type> in a <parameter declaration> shall specify either CHARACTER, BIT, INTEGER, REAL, or DOUBLE PRECISION.

c) If the *i*-th <parameter declaration> specifies a <data type> *PDT* that is CHARACTER(*L*) for some <length> *L*, then the type of the *i*-th parameter *P* shall be Fortran CHARACTER with length equal to the maximum possible length in octets of *PDT*.

d) If the *i*-th <parameter declaration> specifies a <data type> that is BIT(*L*) for some <length> *L*, then the type of the *i*-th parameter is Fortran CHARACTER with length equal to the smallest integer not less than the quotient of the division *L*/*B*, where *B* is the implementation-defined number of bits in a Fortran character.

e) If the *i*-th <parameter declaration> specifies a <data type> that is INTEGER, REAL, or DOUBLE PRECISION, then the type of the *i*-th parameter shall be respectively Fortran INTEGER, REAL, or DOUBLE PRECISION.

5) If the subject <language clause> specifies MUMPS, then:

a) The type of an SQLSTATE parameter shall be MUMPS character with maximum length greater than or equal to 5.

The type of the SQLCODE parameter shall be INTEGER.

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 American Standard. See Annex D, "Deprecated features".

b) Any <data type> in a <parameter declaration> shall specify either CHARACTER VARYING, INTEGER, DECIMAL, or REAL.

c) If the *i*-th <parameter declaration> specifies a <data type> *PDT* that is CHARACTER VARYING(*L*) for some <length> *L*, then the type of the *i*-th parameter shall be MUMPS character with maximum length equal to the maximum possible length in octets of *PDT*.

d) If the *i*-th <parameter declaration> specifies a <data type> *DT* that is INTEGER, DECIMAL, or REAL, then the type of the *i*-th parameter shall be MUMPS character.

6) If the subject <language clause> specifies PASCAL, then:

a) The type of an SQLCODE parameter shall be Pascal INTEGER.

The type of an SQLSTATE parameter shall be Pascal PACKED ARRAY [1..5] OF CHAR.

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 American Standard. See Annex D, "Deprecated features".

b) The argument mode of all parameters shall be VAR.

c) Any <data type> in a <parameter declaration> shall specify either CHARACTER, BIT, INTEGER, or REAL.

d) If the *i*-th <parameter declaration> specifies a <data type> *PDT* that is CHARACTER(*L*) for some <length> *L*, then let *OL* be the maximum possible length in octets of *PDT*. The type of the *i*-th parameter shall be

Case:

1) If *OL* is 1, then Pascal CHAR.

2) Otherwise, Pascal PACKED ARRAY [1..*OL*] OF CHAR.

e) If the *i*-th <parameter declaration> specifies a <data type> that is BIT(*L*) for some <length> *L*, then let *B* be the implementation-defined number of bits in a Pascal character; the type of the *i*-th parameter is

Case:

i) If *L* is between 1 and *B*, inclusive, then Pascal CHAR.

ii) Otherwise, let *BL* be equal to the smallest integer not less than the quotient of the division *L*/*B*; the type is Pascal PACKED ARRAY [1..*BL*] OF CHAR.

f) If the *i*-th <parameter declaration> specifies a <data type> that is INTEGER or REAL, then the type of the *i*-th parameter shall be respectively Pascal INTEGER or Pascal REAL.

7) If the subject <language clause> specifies PLI, then:

a) The type of an SQLCODE parameter shall be PL/I FIXED BINARY (*PP*), where *PP* is an implementation-defined precision that is greater than or equal to 15.

The type of an SQLSTATE parameter shall be PL/I CHARACTER(5).

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 American Standard. See Annex D, "Deprecated features".

b) Any <data type> in a <parameter declaration> shall specify either CHARACTER, CHARACTER VARYING, BIT, BIT VARYING, DECIMAL, INTEGER, SMALLINT, or FLOAT.

c) If the *i*-th <parameter declaration> specifies a <data type> *PDT* that is CHARACTER(*L*) for some <length> *L*, then the type of the *i*-th parameter *P* shall be PL/I CHARACTER with length equal to the maximum possible length in octets of *PDT*.

d) If the *i*-th <parameter declaration> specifies a <data type> *PDT* that is CHARACTER VARYING(*L*) for some <length> *L*, then the type of the *i*-th parameter *P* shall be PL/I CHARACTER VARYING with maximum length equal to maximum possible length in octets of *PDT*.

e) If the *i*-th <parameter declaration> specifies a <data type> that is BIT(*L*) or BIT VARYING(*L*) for some <length> *L*, then the type of the *i*-th parameter is PL/I BIT with length *L* or PL/I BIT VARYING with maximum length *L*, respectively.

f) If the *i*-th <parameter declaration> specifies a <data type> that is DECIMAL(*P*, *S*) for some <precision> and <scale> *P* and *S*, then the type of the *i*-th parameter shall be PL/I FIXED REAL DECIMAL (*P*,*S*).

g) If the *i*-th <parameter declaration> specifies a <data type> that is FLOAT(*P*) for some <precision> *P*, then the type of the *i*-th parameter shall be PL/I FLOAT REAL BINARY (*P*).

h) If the *i*-th <parameter declaration> specifies a <data type> that is INTEGER, then the type of the *i*-th parameter shall be PL/I FIXED BINARY(*PI*), where *PI* is an implementation-defined precision.

i) If the *i*-th <parameter declaration> specifies a <data type> that is SMALLINT, then the type of the *i*-th parameter shall be PL/I FIXED BINARY(*SPI*), where *SPI* is an implementation-defined precision.

Access Rules

None.

General Rules

1) If the subject <language clause> specifies ADA, then:

   a) 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.

   b) 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.

2) If the subject <language clause> specifies C, then:

   Case:

   a) If DT_i specifies BIT(L), then:

      i) 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.

      ii) 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.

   b) If DT_i specifies CHARACTER(L) or CHARACTER VARYING(L), then:

      i) 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:

         1) 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.

         2) 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.

         3) 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.

ii) Let CL_i be one greater than 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, concatenated with a single implementation-defined null character that terminates a C character string.

c) Otherwise,

i) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i 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.

3) If the subject <language clause> specifies COBOL, then:

Case:

a) If DT_i specifies BIT(L), then:

i) 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.

ii) 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.

b) If DT_i specifies CHARACTER(L), then:

i) 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.

ii) 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.

c) Otherwise,

i) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i 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.

4) If the subject <language clause> specifies FORTRAN, then:

Case:

a) If DT_i specifies BIT(L), then:

i) 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.

ii) 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.

b) If DT_i specifies CHARACTER(L), then:

i) 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.

ii) 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.

c) Otherwise,

i) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i 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.

5) If the subject <language clause> specifies MUMPS, then:

Case:

a) If DT_i specifies CHARACTER VARYING(L), then:

i) 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.

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 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.

b) Otherwise, DT_i specifies INT, DEC, or REAL, and:

i) 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)

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

CAST (SV_i AS CHARACTER VARYING(ML))

to PI_i, where ML is the implementation-defined maximum length of variable-length character strings.

6) If the subject <language clause> specifies PASCAL, then:

Case:

a) If DT_i specifies BIT(L), then:

i) 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.

ii) 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.

b) If DT_i specifies CHARACTER(L), then:

i) 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.

ii) 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.

c) Otherwise,

i) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i 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.

7) If the subject <language clause> specifies PLI, then:

Case:

a) If DT_i specifies CHARACTER(L) or CHARACTER VARYING(L), then:

i) 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.

ii) 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:

1) if DT_i specified CHARACTER(L), then an SQL CHARACTER(CL_i) data type

2) 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.

b) Otherwise,

i) Where PN_i is used as an input parameter whose value is evaluated, a reference to PN_i 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL.

None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

None.

12.5 <SQL procedure statement>

Function

Define all of the SQL-statements that are <SQL procedure statement>s.

Format

```
<SQL procedure statement> ::=
         <SQL schema statement>
       | <SQL data statement>
       | <SQL transaction statement>
       | <SQL connection statement>
       | <SQL session statement>
       | <SQL dynamic 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>
       | <domain definition>
       | <character set definition>
       | <collation definition>
       | <translation definition>
       | <assertion definition>

<SQL schema manipulation statement> ::=
         <drop schema statement>
       | <alter table statement>
       | <drop table statement>
       | <drop view statement>
       | <revoke statement>
       | <alter domain statement>
       | <drop domain statement>
       | <drop character set statement>
       | <drop collation statement>
       | <drop translation statement>
       | <drop assertion statement>

<SQL data statement> ::=
         <open statement>
       | <fetch statement>
       | <close statement>
       | <select statement: single row>
       | <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> ::=
```

```
        <set transaction statement>
      | <set constraints mode statement>
      | <commit statement>
      | <rollback statement>

<SQL connection statement> ::=
        <connect statement>
      | <set connection statement>
      | <disconnect statement>

<SQL session statement> ::=
        <set catalog statement>
      | <set schema statement>
      | <set names statement>
      | <set session authorization identifier statement>
      | <set local time zone statement>

<SQL dynamic statement> ::=
        <system descriptor statement>
      | <prepare statement>
      | <deallocate prepared statement>
      | <describe statement>
      | <execute statement>
      | <execute immediate statement>
      | <SQL dynamic data statement>

<SQL dynamic data statement> ::=
        <allocate cursor statement>
      | <dynamic open statement>
      | <dynamic fetch statement>
      | <dynamic close statement>
      | <dynamic delete statement: positioned>
      | <dynamic update statement: positioned>

<system descriptor statement> ::=
        <allocate descriptor statement>
      | <deallocate descriptor statement>
      | <set descriptor statement>
      | <get descriptor statement>

<SQL diagnostics statement> ::=
      <get diagnostics statement>
```

Syntax Rules

None.

Access Rules

None.

General Rules

None.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

13 Data manipulation

13.1 <declare cursor>

Function

Define a cursor.

Format

```
<declare cursor> ::=
       DECLARE <cursor name> [ INSENSITIVE ] [ SCROLL ] CURSOR
         FOR <cursor specification>

<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> ::=
         <column name>
       | <unsigned integer>

<ordering specification> ::= ASC | DESC
```

Syntax Rules

1) The <cursor name> shall be different from the <cursor name> contained in any other <declare cursor> in the same <module>.

2) Any <parameter name> contained in the <cursor specification> shall be defined in a <parameter declaration> in the <procedure> in the containing <module> that contains an <open statement> that specifies the <cursor name>.

   Note: See the Syntax Rules of Subclause 12.1, "<module>".

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 a read-only 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 a *read-only cursor*; otherwise, *CS* is an *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) The *simply underlying table* of a <cursor specification> is the table derived from by the <query expression> simply contained in the <cursor specification>.

9) If *T* is an updatable table, then let *TU* be the leaf generally underlying table of the <cursor specification>.

10) If ORDER BY is specified, then each <sort specification> in the <order by clause> shall identify a column of *T*.

    Case:

    a) If a <sort specification> contains a <column name>, then *T* shall contain exactly one column with that <column name> and the <sort specification> identifies that column.

    b) If a <sort specification> contains an <unsigned integer>, then the <unsigned integer> shall be greater than 0 and not greater than the degree of *T*. The <sort specification> identifies the column of *T* with the ordinal position specified by the <unsigned integer>.

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 or implicit, then *T* shall be an updatable table and INSENSITIVE shall not be specified.

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 *TU* is 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 *TU*.

Access Rules

None.

General Rules

1) If *T* is an 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.

2) 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.

3) 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   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.

2) 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.

Format

```
<open statement> ::=
       OPEN <cursor name>
```

Syntax Rules

1) The containing <module> shall contain a <declare cursor> *CR* whose <cursor name> is the same as the <cursor name> in the <open statement>.

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 cursor *CR* is not in the closed state, then an exception condition is raised: *invalid cursor state*.

2) Let *S* be the <cursor specification> of cursor *CR*.

3) 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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

None.

2) 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) The containing <module> shall contain a <declare cursor> *CR* whose <cursor name> is the same as the <cursor name> in the <fetch statement>. Let *T* be the table defined by the <cursor specification> of *CR*.

3) If the implicit or explicit <fetch orientation> is not NEXT, then the <declare cursor> *CR* 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) 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*.

6) 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) 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.

3) 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.

4) 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.

5) 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.

6) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

7) 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.

8) 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.

9) If an exception condition occurs during the assignment of *SV* to *TV*, then the value of *TV* is implementation-dependent and *CR* remains positioned on the current row.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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

1) The containing <module> shall contain a <declare cursor> *CR* whose <cursor name> is the same as the <cursor name> of the <close statement>.

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) Cursor *CR* is placed in the closed state and the copy of the <cursor specification> of *CR* is destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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) 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>.

2) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to each corresponding <target specification> and <value expression>, as *TARGET* and *VALUE*, respectively.

3) 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) The assignment of values to targets in the <select target list> is in an implementation-dependent order.

5) If an exception condition is raised during the assignment of a value to a target, then the values of targets are implementation-dependent.

6) 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*.

7) Let *TV* be an identified target and let *SV* be its corresponding value in the row of *Q*.

8) The General Rules of Subclause 9.1, "Retrieval assignment", are applied to *TV* and *SV*, as *TARGET* and *VALUE*, respectively.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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> ::=
       DELETE FROM <table name>
         WHERE CURRENT OF <cursor name>
```

Syntax Rules

1) The containing <module> shall contain a <declare cursor> whose <cursor name> is the same as the <cursor name> in the <delete statement: positioned>. Let *CR* be the cursor specified by <cursor name>.

2) *CR* shall be an updatable cursor.
   Note: *updatable cursor* is defined in Subclause 13.1, "<declare cursor>".

3) Let *T* be the table identified by the <table name>. 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>".

Access Rules

1) The applicable privileges shall include DELETE for the <table name>.
   Note: The *applicable privileges* for a <table name> are defined in Subclause 10.3, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) If cursor *CR* is not positioned on a row, then an exception condition is raised: *invalid cursor state*.

3) The row from which the current row of *CR* is derived is marked for deletion.

4) 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*.

5) 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.

6) 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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 name>
         [ WHERE <search condition> ]
```

Syntax Rules

1) Let *T* be the table identified by the <table name>. *T* shall not be a read-only table.

2) The scope of the <table name> is the entire <delete statement: searched>.

Access Rules

1) The applicable privileges shall include DELETE for the <table name>.
   Note: The *applicable privileges* for a <table name> are defined in Subclause 10.3, "<privileges>".

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) Case:

   a) If <search condition> is not specified, then all rows of *T* are marked for deletion.

   b) If <search condition> is specified, then it is applied to each row of *T* with the <table name> bound to that row, and all rows for which the result of the <search condition> is true are marked for deletion.

      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 6.4, "<column reference>".

3) 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*.

4) 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 constraint.

5) If no row is deleted, then a completion condition is raised: *no data*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) No leaf generally underlying table of *T* shall be an underlying table of any <query expression> generally contained in the <search condition>.

2) 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 name>
         <insert columns and source>

<insert columns and source> ::=
         [ <left paren> <insert column list> <right paren> ] <query expression>
       | DEFAULT VALUES

<insert column list> ::= <column name list>
```

Syntax Rules

1) The table *T* identified by the <table name> shall not be a read-only table.

2) An <insert columns and source> that specifies DEFAULT VALUES is equivalent to an <insert columns and source> that specifies a <query expression> of the form

   VALUES (DEFAULT, . . .)

   where the number of "DEFAULT" entries is equal to the number of columns of *T*.

3) No <column name> of *T* shall be identified more than once. If 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.

4) A column identified by the <insert column list> is an object column.

5) 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*.

6) The Syntax Rules of Subclause 9.2, "Store assignment", apply to corresponding columns of *T* and *QT* as *TARGET* and *VALUE*, respectively.

Access Rules

1) Case:

   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*.

   Note: The *applicable privileges* for a <table name> are defined in Subclause 10.3, "<privileges>".

2) Each <column name> in the <insert column list> shall identify a column of *T*.

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) Let *B* be the leaf generally underlying table of *T*.

3) The <query expression> is effectively evaluated before inserting any rows into *B*.

4) 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 11.5, "<default clause>". The candidate row includes every column of *B*.

      ii) 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*.

      iii) 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.

      iv) The candidate row is inserted into *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 11.19, "<view definition>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   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 <qualifier> of a <column reference>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) The <query expression> that is contained in an <insert statement> 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>.

   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> ::=
       UPDATE <table name>
         SET <set clause list>
           WHERE CURRENT OF <cursor name>

<set clause list> ::=
       <set clause> [ { <comma> <set clause> }... ]

<set clause> ::=
       <object column> <equals operator> <update source>

<update source> ::=
         <value expression>
       | <null specification>
       | DEFAULT

<object column> ::= <column name>
```

Syntax Rules

1) The containing <module> shall contain a <declare cursor> for a cursor whose <cursor name> is the same as the <cursor name> in the <update statement: positioned>. Let *CR* be the cursor specified by <cursor name>.

2) *CR* shall be an updatable cursor.

   Note: *updatable cursor* is defined in Subclause 13.1, "<declare cursor>".

3) Let *T* be the table identified by the <table name>. 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>".

4) 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*.

5) 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>.

6) A <value expression> in a <set clause> shall not directly contain a <set function specification>.

7) The same <object column> shall not appear more than once in a <set clause list>.

8) 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>.

9) The scope of the <table name> is the entire <update statement: positioned>.

10) For every <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.

Access Rules

1) The applicable privileges shall include UPDATE for each <object column>.

   Note: The *applicable privileges* for a <table name> are defined in Subclause 10.3, "<privileges>".

2) Each <column name> specified as an <object column> shall identify a column of *T*.

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) If cursor *CR* is not positioned on a row, then an exception condition is raised: *invalid cursor state*.

3) The object row is that row from which the current row of *CR* is derived.

4) If, while *CR* is open, 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*.

5) The value of DEFAULT is the default value indicated in the column descriptor for the <object column> in the containing <set clause>.

6) The <value expression>s are effectively evaluated before updating the object row. If a <value expression> contains a reference to a column of *T*, then the reference is to the value of that column in the object row before any value of the object row is updated.

7) *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.

8) 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 11.19, "<view definition>".

9) 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) *CR* shall not be an ordered cursor.

2) 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 name>
         SET <set clause list>
         [ WHERE <search condition> ]
```

Syntax Rules

1) Let *T* be the table identified by the <table name>. *T* shall be an updatable table.

2) A <value expression> in a <set clause> shall not directly contain a <set function specification>.

3) The same <object column> shall not appear more than once in a <set clause list>.

4) The scope of the <table name> is the entire <update statement: searched>.

5) For every <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.

Access Rules

1) The applicable privileges shall include UPDATE for each <object column>.

   Note: The *applicable privileges* for a <table name> are defined in Subclause 10.3, "<privileges>".

2) Each <column name> specified as an <object column> shall identify a column of *T*.

General Rules

1) If the access mode of the current SQL-transaction is *read-only* and *T* is not a temporary table, then an exception condition is raised: *invalid transaction state*.

2) Case:

   a) If a <search condition> is not specified, then all rows of *T* are the 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 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*, the reference is to the value of that column in the given row of *T*.

      Note: *Outer reference* is defined in Subclause 6.4, "<column reference>".

3) 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*.

4) If the set of object rows is empty, then a completion condition is raised: *no data*.

5) If a completion condition *no data* has been raised, then no further General Rules of this Subclause are applied.

6) The <value expression>s are effectively evaluated for each row of *T* before updating any row of *T*.

7) 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 11.19, "<view definition>".

8) Each 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.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   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>.

2) 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*.

13.11 <temporary table declaration>

Function

Declare a declared local temporary table that will be effectively materialized the first time that any <procedure> in the <module> that contains the <temporary table declaration> is executed and whose scope is all the <procedure>s of that <module> executed within the same SQL-session.

Format

```
<temporary table declaration> ::=
       DECLARE LOCAL TEMPORARY TABLE <qualified local table name>
         <table element list>
         [ ON COMMIT { PRESERVE | DELETE } ROWS ]
```

Syntax Rules

1) Let *T* be the <local table name> of <qualified local table name>. *T* shall be different from the <local table name> of any other <temporary table declaration> contained within the <module>.

2) Let *A* be the current <authorization identifier>.

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 (<column name>), UPDATE (<column name>), and REFERENCES (<column name>) 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*

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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <temporary table declaration>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14 Transaction management

14.1 <set transaction statement>

Function

Set the attributes of the next SQL-transaction for the SQL-agent.

Format

```
<set transaction statement> ::=
       SET 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 <set transaction statement> statement is executed when an SQL-transaction is currently active, then an exception condition is raised: *invalid transaction state*.

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 next SQL-transaction for the SQL-agent.

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 9, "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 Intermediate SQL:

   None.

2) 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>.

14.2 <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.

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, 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*.

3) 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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <set constraints mode statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

14.3 <commit statement>

Function

Terminate the current SQL-transaction with commit.

Format

```
<commit statement> ::=
       COMMIT [ WORK ]
```

Syntax Rules

None.

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) For every open cursor *CR* in any <module> associated with the current SQL-transaction, the following statement is implicitly executed:

   CLOSE *CR*

3) 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 execution of a <delete statement: searched> that specifies DELETE FROM *T*, where *T* is the <table name> of that temporary table.

4) The effects specified in the General Rules of Subclause 14.2, "<set constraints mode statement>" occur as if the statement SET CONSTRAINTS ALL IMMEDIATE were executed.

5) 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 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.

   c) 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.

6) The current SQL-transaction is terminated.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) In conforming Entry SQL language, WORK shall be specified.

14.4 <rollback statement>

Function

Terminate the current SQL-transaction with rollback.

Format

```
<rollback statement> ::=
       ROLLBACK [ WORK ]
```

Syntax Rules

None.

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) For every open cursor *CR* in any <module> associated with the current SQL-transaction, the following statement is implicitly executed:

   CLOSE *CR*

3) Any changes to SQL-data or schemas that were made by the current SQL-transaction are canceled.

4) The current SQL-transaction is terminated.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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-connection.

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 *CV*)

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 <procedure> 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-server 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 4.30, "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 <procedure>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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <connect statement>.

2) 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-server 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 4.30, "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 current 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 4.30, "SQL-sessions".

6) The SQL-server for the subsequent execution of <procedure>s in any <module>s associated with the SQL-agent is set to that of the current SQL-connection.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <set connection statement>.

2) 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 any. 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*.

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 <procedure> 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 12.1, "<module>", for the actions to be performed after the last call of a <procedure> 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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <disconnect statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

16 Session management

16.1 <set catalog statement>

Function

Set the default catalog name for unqualified <schema name>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set catalog statement> ::=
      SET CATALOG <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) 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*)

2) If *V* does not conform to the Format and Syntax Rules of a <catalog name>, then an exception condition is raised: *invalid catalog name*.

3) The default catalog name of the current SQL-session is set to *V*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <set catalog statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions

   None.

16.2 <set schema statement>

Function

Set the default schema name for unqualified <qualified name>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set schema statement> ::=
       SET SCHEMA <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) 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*)

2) If *V* does not conform to the Format and Syntax Rules of a <schema name>, then an exception condition is raised: *invalid schema name.*

3) Case:

   a) If *V* conforms to the Format and Syntax Rules for a <schema name> that contains a <catalog name>, then let *X* be the <catalog name> part and let *Y* be the <unqualified schema name> part of *V*. The following statement is implicitly executed:

      SET CATALOG '*X*'

      and the <set schema statement> is effectively replaced by:

      SET SCHEMA '*Y*'

   b) Otherwise, the default unqualified schema name of the current SQL-session is set to *V*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <set schema statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

16.3 <set names statement>

Function

Set the default character set name for <identifier>s and <character string literal>s in <preparable statement>s that are prepared in the current SQL-session by an <execute immediate statement> or a <prepare statement> and in <direct SQL statement>s that are invoked directly.

Format

```
<set names statement> ::=
       SET NAMES <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) 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*)

2) If *V* does not conform to the Format and Syntax Rules of a <character set name>, then an exception condition is raised: *invalid character set name*.

3) The default character set name of the current SQL-session is set to *V*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <set names statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

16.4 <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*.

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 defined in 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*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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>.

16.5 <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 Intermediate SQL;

   None.

2) The following restrictions apply for Entry SQL;

   a) Conforming Entry SQL language shall not contain any <set local time zone statement>.

17 Dynamic SQL

17.1 Description of SQL item descriptor areas

Function

Specify the identifiers, data types, and codes used in SQL item descriptor areas.

Syntax Rules

1) An SQL item descriptor area comprises the items specified in Table 17, "Data types of <key word>s used in SQL item descriptor areas".

2) Let *DT* be a data type. The data type *T* of a <simple value specification> or a <simple target specification> *SVT* is said to *match* the data type specified by the item descriptor area if and only if one of the following conditions is true.

   Case:

   a) TYPE indicates NUMERIC and *T* is specified by NUMERIC(*P*,*S*), where *P* is the value of PRECISION and *S* is the value of SCALE.

   b) TYPE indicates DECIMAL and *T* is specified by DECIMAL(*P*,*S*), where *P* is the value of PRECISION and *S* is the value of SCALE.

   c) TYPE indicates INTEGER and *T* is specified by INTEGER.

   d) TYPE indicates SMALLINT and *T* is specified by SMALLINT.

   e) TYPE indicates FLOAT and *T* is specified by FLOAT(*P*), where *P* is the value of PRECISION.

   f) TYPE indicates REAL and *T* is specified by REAL.

   g) TYPE indicates DOUBLE PRECISION and *T* is specified by DOUBLE PRECISION.

   h) TYPE indicates BIT and *T* is specified by BIT(*L*), where *L* is the value of LENGTH.

   i) TYPE indicates BIT VARYING and *T* is specified by BIT VARYING(*L*), where

      Case:

      i) *SVT* is a <simple value specification> and *L* is the value of LENGTH.

      ii) *SVT* is a <simple target specification> and *L* is not less than the value of LENGTH.

   j) TYPE indicates CHARACTER and *T* is specified by CHARACTER(*L*), where *L* is the value of LENGTH and the <character set specification> formed by the values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME identifies the character set of *SVT*.

k) TYPE indicates CHARACTER VARYING and *T* is specified by CHARACTER VARYING(*L*), where the <character set specification> formed by the values of CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME identifies the character set of *SVT* and

Case:

i) *SVT* is a <simple value specification> and *L* is the value of LENGTH.

ii) *SVT* is a <simple target specification> and *L* is not less than the value of LENGTH.

3) An item descriptor area is *valid* if and only if TYPE indicates a code defined in Table 18, "Codes used for SQL data types in Dynamic SQL", and one of the following is true:

Case:

a) TYPE indicates NUMERIC and PRECISION and SCALE are valid precision and scale values for the NUMERIC data type.

b) TYPE indicates DECIMAL and PRECISION and SCALE are valid precision and scale values for the DECIMAL data type.

c) TYPE indicates FLOAT and PRECISION is a valid precision value for the FLOAT data type.

d) TYPE indicates INTEGER, SMALLINT, REAL, or DOUBLE PRECISION.

e) TYPE indicates BIT or BIT VARYING and LENGTH is a valid length value for the BIT date type.

f) TYPE indicates CHARACTER or CHARACTER VARYING, LENGTH is a valid length value for the CHARACTER data type, and CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are a valid qualified character set name for the CHARACTER data type.

g) TYPE indicates a <datetime type>, DATETIME_INTERVAL_CODE is a code specified in Table 19, "Codes associated with datetime data types in Dynamic SQL", and PRECISION is a valid value for the <time precision> or <timestamp precision> of the indicated datetime datatype.

h) TYPE indicates an <interval type>, DATETIME_INTERVAL_CODE is a code specified in Table 20, "Codes used for <interval qualifier>s in Dynamic SQL", and DATETIME_INTERVAL_PRECISION and PRECISION are valid values for <interval leading field precision> and <interval fractional seconds precision> for an <interval qualifier>.

Table 17—Data types of <key word>s used in SQL item descriptor areas

<key word>	Data Type
TYPE	exact numeric with scale 0
LENGTH	exact numeric with scale 0
OCTET_LENGTH	exact numeric with scale 0

Table 17—Data types of <key word>s used in SQL item descriptor areas (Cont.)

<key word>	Data Type
RETURNED_LENGTH	exact numeric with scale 0
RETURNED_OCTET_LENGTH	exact numeric with scale 0
PRECISION	exact numeric with scale 0
SCALE	exact numeric with scale 0
DATETIME_INTERVAL_CODE	exact numeric with scale 0
DATETIME_INTERVAL_ PRECISION	exact numeric with scale 0
NULLABLE	exact numeric with scale 0
INDICATOR	exact numeric with scale 0
DATA	matches data type specified by TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_ PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_ SCHEMA, and CHARACTER_SET_NAME
NAME	character string with character set SQL_TEXT and length not less than 128 characters
UNNAMED	exact numeric with scale 0
COLLATION_CATALOG	character string with character set SQL_TEXT and length not less than 128 characters
COLLATION_SCHEMA	character string with character set SQL_TEXT and length not less than 128 characters
COLLATION_NAME	character string with character set SQL_TEXT and length not less than 128 characters
CHARACTER_SET_CATALOG	character string with character set SQL_TEXT and length not less than 128 characters
CHARACTER_SET_SCHEMA	character string with character set SQL_TEXT and length not less than 128 characters
CHARACTER_SET_NAME	character string with character set SQL_TEXT and length not less than 128 characters

Access Rules

None.

General Rules

1) Table 18, "Codes used for SQL data types in Dynamic SQL", specifies the codes associated with the SQL data types.

Table 18—Codes used for SQL data types in Dynamic SQL

Data Type	Code
Implementation-defined data types	< 0
BIT	14
BIT VARYING	15
CHARACTER	1
CHARACTER VARYING	12
DATE, TIME, or TIMESTAMP	9
DECIMAL	3
DOUBLE PRECISION	8
FLOAT	6
INTEGER	4
INTERVAL	10
NUMERIC	2
REAL	7
SMALLINT	5

2) Table 19, "Codes associated with datetime data types in Dynamic SQL", specifies the codes associated with the datetime data types.

Table 19—Codes associated with datetime data types in Dynamic SQL

Datetime Data Type	Code
DATE	1
TIME	2
TIME WITH TIME ZONE	4
TIMESTAMP	3
TIMESTAMP WITH TIME ZONE	5

3) Table 20, "Codes used for <interval qualifier>s in Dynamic SQL", specifies the codes associated with <interval qualifier>s for interval data types.

Table 20—Codes used for <interval qualifier>s in Dynamic SQL

Datetime Qualifier	Code
DAY	3
DAY TO HOUR	8
DAY TO MINUTE	9
DAY TO SECOND	10
HOUR	4
HOUR TO MINUTE	11
HOUR TO SECOND	12
MINUTE	5
MINUTE TO SECOND	13
MONTH	2
SECOND	6
YEAR	1
YEAR TO MONTH	7

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

17.2 <allocate descriptor statement>

Function

Allocate an SQL descriptor area.

Format

```
<allocate descriptor statement> ::=
       ALLOCATE DESCRIPTOR <descriptor name> [ WITH MAX <occurrences> ]

<occurrences> ::= <simple value specification>
```

Syntax Rules

1) The data type of <occurrences> shall be exact numeric with scale 0.

2) If WITH MAX <occurrences> is not specified, then an implementation-defined default value for <occurrences> that is greater than 0 is implicit.

Access Rules

None.

General Rules

1) Let *S* be the character string that is the value of the <simple value specification> that is immediately contained in <descriptor name> and let *V* be the character string that is the result of

   TRIM (BOTH ' ' FROM *S*)

   If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid SQL descriptor name*.

2) The <allocate descriptor statement> allocates an SQL descriptor area whose name is *V* and whose scope is specified by the <scope clause>. The descriptor area will have at least <occurrences> number of item descriptor areas. All values are initially undefined. If an SQL descriptor has already been allocated whose name is *V*, whose scope is specified by the <scope option>, and that has not yet been deallocated, then an exception condition is raised: *invalid SQL descriptor name*.

3) If <occurrences> is less than 1 or is greater than an implementation-defined maximum value, then an exception condition is raised: *dynamic SQL error—invalid descriptor index*. The maximum number of SQL descriptor areas that can be allocated at one time is implementation-defined.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) An <occurrences> and a <descriptor name> shall be a <literal>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.3 <deallocate descriptor statement>

Function

Deallocate an SQL descriptor area.

Format

```
<deallocate descriptor statement> ::=
       DEALLOCATE DESCRIPTOR <descriptor name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) The <deallocate descriptor statement> deallocates an SQL descriptor area whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is specified by the <scope clause>. If an SQL descriptor area is not currently allocated whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is specified by the <scope option>, then an exception condition is raised: *invalid SQL descriptor name*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <descriptor name> shall be a <literal>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall contain no Dynamic SQL language.

17.4 <get descriptor statement>

Function

Get information from an SQL descriptor area.

Format

```
<get descriptor statement> ::=
       GET DESCRIPTOR <descriptor name> <get descriptor information>

<get descriptor information> ::=
         <get count>
       | VALUE <item number>
           <get item information> [ { <comma> <get item information> }... ]

<get count> ::=
       <simple target specification 1> <equals operator> COUNT

<get item information> ::=
       <simple target specification 2> <equals operator> <descriptor item name>>

<item number> ::= <simple value specification>

<simple target specification 1> ::= <simple target specification>

<simple target specification 2> ::= <simple target specification>

<descriptor item name> ::=
         TYPE
       | LENGTH
       | OCTET_LENGTH
       | RETURNED_LENGTH
       | RETURNED_OCTET_LENGTH
       | PRECISION
       | SCALE
       | DATETIME_INTERVAL_CODE
       | DATETIME_INTERVAL_PRECISION
       | NULLABLE
       | INDICATOR
       | DATA
       | NAME
       | UNNAMED
       | COLLATION_CATALOG
       | COLLATION_SCHEMA
       | COLLATION_NAME
       | CHARACTER_SET_CATALOG
       | CHARACTER_SET_SCHEMA
       | CHARACTER_SET_NAME
```

Syntax Rules

1) The data type of <item number> shall be exact numeric with scale 0.

2) The data type of <simple target specification 1> shall be exact numeric with scale 0.

3) The data type of a <simple target specification 2> shall be the data type shown in Table 17, "Data types of <key word>s used in SQL item descriptor areas", for the corresponding <descriptor item name>.

Access Rules

None.

General Rules

1) If a <descriptor name> specified in a <get descriptor statement> identifies an SQL descriptor area that is not currently allocated, then an exception condition is raised: *invalid SQL descriptor name*.

2) If the data type of the <simple target specification> associated with the keyword DATA does not match the data type specified by the item descriptor area, then an exception condition is raised: *data exception—error in assignment*.

   Note: *Match* is defined in the Syntax Rules of Subclause 17.1, "Description of SQL item descriptor areas".

3) If the <item number> specified in a <get descriptor statement> is greater than the number of <occurrences> specified when the <descriptor name> was allocated or less than 1, then an exception condition is raised: *dynamic SQL error—invalid descriptor index*.

4) If the <item number> specified in a <get descriptor statement> is greater than the value of COUNT, then a completion condition is raised: *no data*.

5) If a <get descriptor statement> is executed to get the value of DATA without getting the value of INDICATOR and the value of INDICATOR is negative, then an exception condition is raised: *data exception—null value, no indicator parameter*.

6) If an exception condition is raised in a <get descriptor statement>, then the values of all targets specified by <simple target specification 1> and <simple target specification 2> are implementation-dependent.

7) A <get descriptor statement> gets values in the SQL descriptor area specified by <descriptor name>. The values are as follows:

   a) COUNT is a count of the number of <dynamic parameter specification>s or <select list> columns described in item descriptor areas.

   b) The values of TYPE, LENGTH, OCTET_LENGTH, RETURNED_LENGTH, RETURNED_OCTET_LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are the data type, length (in characters, bits, or positions, as appropriate), length in octets, returned length in characters, returned length in octets, precision, scale, datetime data type, interval qualifier, collation, and character set of a <dynamic parameter specification> or <select list> column described by the item descriptor area specified by <item number>.

c) NAME is the name associated with the <dynamic parameter specification> or <select list> column described by the item descriptor area specified by <item number>. For a <select list> column, if the column name is implementation-dependent, then NAME is the implementation-dependent name for the column and UNNAMED is set to 1; otherwise, NAME is the <derived column name> of the column and UNNAMED is set to 0. For a <dynamic parameter specification>, the values of NAME and UNNAMED are implementation-dependent.

d) DATA is a value for the <target specification> described by the item descriptor area specified by <item number>. If the value of INDICATOR is negative, then the value of DATA is undefined.

e) INDICATOR is a value for the <indicator parameter> associated with the <target specification>.

f) Case:

i) For a <select list> column, if NULLABLE is set to 1, then the column can have the null value; otherwise, the column cannot have the null value.

ii) For a <dynamic parameter specification>, NULLABLE is set to 1, indicating that the <dynamic parameter specification> can have the null value.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

a) A <descriptor name> shall be a <literal>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.5 <set descriptor statement>

Function

Set information in an SQL descriptor area.

Format

```
<set descriptor statement> ::=
       SET DESCRIPTOR <descriptor name> <set descriptor information>

<set descriptor information> ::=
         <set count>
       | VALUE <item number>
           <set item information> [ { <comma> <set item information> }... ]

<set count> ::=
       COUNT <equals operator> <simple value specification 1>

<set item information> ::=
       <descriptor item name> <equals operator> <simple value specification 2>

<simple value specification 1> ::= <simple value specification>

<simple value specification 2> ::= <simple value specification>

<item number> ::= <simple value specification>
```

Syntax Rules

1) The data type of <item number> shall be exact numeric with scale 0.

2) The data type of <simple value specification 1> shall be an exact numeric with scale 0.

3) The data type of a <simple value specification 2> shall be the data type shown in Table 17, "Data types of <key word>s used in SQL item descriptor areas", for the corresponding <descriptor item name>. <descriptor item name> shall not be RETURNED_LENGTH, RETURNED_OCTET_LENGTH, OCTET_LENGTH, NULLABLE, COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME, NAME, or UNNAMED. Other alternatives for <descriptor item name> shall not be specified more than once in a <set descriptor statement>.

Access Rules

None.

General Rules

1) If a <descriptor name> specified in a <set descriptor statement> identifies an SQL descriptor area that is not currently allocated, then an exception condition is raised: *invalid SQL descriptor name*.

2) If the <item number> specified in a <set descriptor statement> is greater than the number of <occurrences> specified when the <descriptor name> was allocated or less than 1, then an exception condition is raised: *dynamic SQL error—invalid descriptor index*.

3) A <set descriptor statement> sets values in the SQL descriptor area specified by <descriptor name>. The values are as follows:

   a) COUNT is a count of the number of <dynamic parameter specification> values or <target specification>s described in item descriptor areas.

   b) TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are a description of the data type, length (in characters, bits, or positions, as appropriate), precision, scale, datetime data type, interval qualifier, and character set of a <dynamic parameter specification> value or <target specification> described by the item descriptor area specified by <item number>.

   c) DATA is a value for the <dynamic parameter specification> described by the item descriptor area specified by <item number>. INDICATOR is a value for the <indicator parameter> associated with the <dynamic parameter specification>.

4) When more than one value is set in a single <set descriptor statement>, the values are effectively assigned in the following order: TYPE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, CHARACTER_SET_NAME, LENGTH, INDICATOR, and DATA.

   When any value other than DATA is set, the value of DATA becomes undefined.

5) For every <set item information> specified, let *DIN* be the <descriptor item name> and let *V* be the value of the <simple value specification 2>. If *DIN* is DATA and the data type of *V* does not match the data type specified by the item descriptor area, then an exception condition is raised: *data exception—error in assignment*.

   Note: *Match* is defined in the Syntax Rules of Subclause 17.1, "Description of SQL item descriptor areas".

   The item descriptor area field *DIN* is set to *V* and then

   Case:

   a) If *DIN* is TYPE and *V* indicates CHARACTER or CHARACTER VARYING, then CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME are set to the values for the default character set name for the SQL-session, LENGTH is set to 1, and all other item descriptor area fields are set to implementation-dependent values.

   b) If *DIN* is TYPE and *V* indicates BIT or BIT VARYING, then LENGTH is set to 1 and all other item descriptor area fields are set to implementation-dependent values.

   c) If *DIN* is TYPE and *V* indicates DATETIME, then PRECISION is set to 0 and all other item descriptor area fields are set to implementation-dependent values.

   d) If *DIN* is TYPE and *V* indicates INTERVAL, then DATETIME_INTERVAL_PRECISION is set to 2 and all other item descriptor area fields are set to implementation-dependent values.

   e) If *DIN* is TYPE and *V* indicates NUMERIC or DECIMAL, then SCALE is set to 0, PRECISION is set to the implementation-defined default value for the precision of NUMERIC or DECIMAL data types, respectively, and all other item descriptor area fields are set to implementation-dependent values.

   f) If *DIN* is TYPE and *V* indicates INTEGER or SMALLINT, then all other item descriptor area fields are set to implementation-dependent values.

g) If *DIN* is TYPE and *V* indicates FLOAT, then PRECISION is set to the implementation-defined default value for the precision of FLOAT data types and all other item descriptor area fields are set to implementation-dependent values.

h) If *DIN* is TYPE and *V* indicates REAL or DOUBLE PRECISION, then all other item descriptor area fields are set to implementation-dependent values.

i) If *DIN* is DATETIME_INTERVAL_CODE and TYPE is DATETIME, then

i) If *V* indicates DATE, TIME, or TIME WITH TIME ZONE, then PRECISION is set to 0 and all other item descriptor area fields are set to implementation-dependent values.

ii) If *V* indicates TIMESTAMP, or TIMESTAMP WITH TIME ZONE, then PRECISION is set to 6 and all other item descriptor area fields are set to implementation-dependent values.

j) If *DIN* is DATETIME_INTERVAL_CODE and TYPE is INTERVAL, then DATETIME_INTERVAL_PRECISION is set to 2 and

i) If *V* indicates DAY TO SECOND, HOUR TO SECOND, MINUTE TO SECOND, or SECOND, then PRECISION is set to 6.

ii) Otherwise, PRECISION is set to 0.

6) If an exception condition is raised in a <set descriptor statement>, then the values of all elements of the item descriptor area specified in the <set descriptor statement> are implementation-dependent.

7) Restrictions on changing TYPE, LENGTH, PRECISION, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME values resulting from the execution of a <describe statement> before execution of a <execute statement>, <dynamic open statement>, or <dynamic fetch statement> are implementation-defined, except as specified in the General Rules of Subclause 17.9, "<using clause>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

a) A <descriptor name> shall be a <literal>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.6 <prepare statement>

Function

Prepare a statement for execution.

Format

```
<prepare statement> ::=
       PREPARE <SQL statement name> FROM <SQL statement variable>

<SQL statement variable> ::= <simple value specification>

<preparable statement> ::=
         <preparable SQL data statement>
       | <preparable SQL schema statement>
       | <preparable SQL transaction statement>
       | <preparable SQL session statement>
       | <preparable implementation-defined statement>

<preparable SQL data statement> ::=
         <delete statement: searched>
       | <dynamic single row select statement>
       | <insert statement>
       | <dynamic select statement>
       | <update statement: searched>
       | <preparable dynamic delete statement: positioned>
       | <preparable dynamic update statement: positioned>

<preparable SQL schema statement> ::=
         <SQL schema statement>

<preparable SQL transaction statement> ::=
         <SQL transaction statement>

<preparable SQL session statement> ::=
         <SQL session statement>

<dynamic select statement> ::= <cursor specification>

<dynamic single row select statement> ::= <query specification>

<preparable implementation-defined statement> ::= !! See the Syntax Rules.
```

Syntax Rules

1) The <simple value specification> of <SQL statement variable> shall not be a <literal>.

2) The data type of <SQL statement variable> shall be character string.

3) The Format and Syntax Rules for <preparable implementation-defined statement> are implementation-defined.

Access Rules

None.

General Rules

1) Let *P* be the contents of the <SQL statement variable>.

2) If *P* is a <preparable dynamic delete statement: positioned> or a <preparable dynamic update statement: positioned>, then *P* refers to either a dynamic cursor with the same <cursor name> or to an extended dynamic cursor whose <extended cursor name> value is the same as the <cursor name>.

   Case:

   a) If both an extended dynamic cursor and a dynamic cursor with the same name as the <cursor name> exist, then an exception condition is raised: *ambiguous cursor name*.

   b) If there is neither an extended dynamic cursor nor a dynamic cursor with the name of <cursor name>, then an exception condition is raised: *invalid cursor name*.

3) Let *E1*, *E2*, *E3*, and *E4* be <value expression>s of the form "<dynamic parameter specification>" or a <dynamic parameter specification> enclosed in any number of matching parentheses. Let *F* be a <value expression> whose data type is determined via General Rules in Clause 6, "Scalar expressions" and the Rules in this Subclause. Let *X1*, *X2*, *X3*, and *X4* be <value expression>s meeting the criteria for either *En* or *F*.

4) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation in dynamic SQL statement*.

   a) *P* does not conform to the Format, Syntax Rules, and Access Rules of a <preparable statement>.

   b) *P* contains *E1* as a <value expression> simply contained in a <select list>.

   c) *P* contains *E1* as both operands of a single dyadic operator.

   d) *P* contains a <value expression> of the form "+ *E1*", "– *E1*", "*E1* COLLATE <collation name>", or "EXTRACT (<extract field> FROM *E1*)".

   e) *P* contains a <set function specification> with argument *E1*.

   f) *P* contains a <null predicate> with a value *E1* in the <row value constructor>.

   g) *P* contains an <overlaps predicate> where *E1* is the <value expression> that is the second <row value constructor element> in either the first or second operands of the <overlaps predicate>.

   h) *P* contains *E1* as the first operand of COALESCE or the first <when operand> in a <case specification> or both operands of NULLIF.

   i) *P* contains a <comparison predicate> or <between predicate> where *E1* is both the *i*-th <value expression> in the <row value constructor> that is the first operand and the *i*-th <value expression> in the <row value constructor> that is the second operand of the <comparison predicate> or the second or third operand of the <between predicate>.

j) *P* contains a <table value constructor> in which the *i*-th <value expression> in each <row value constructor> is *E1* and either:

i) *P* is not an <insert statement>, or

ii) *P* is an <insert statement> and the <table value constructor> is not the <query expression> simply contained in the <insert statement>.

k) *P* contains an <in predicate> with an <in value list> in which *E1* is both the <row value constructor> and the first <value specification> of the <in value list>.

l) *P* contains a <position expression> in which both immediately contained <character value expression>s are *E1*.

m) *P* contains a <form-of-use conversion> or a <character translation> whose immediately contained <character value expression> is *E1*.

n) *P* contains a <fold> whose operand is *E1*.

o) *P* contains a <trim function> whose <trim character> or <trim source> is *E1*.

p) *P* contains a <character substring function> whose <character value expression> is *E1*.

q) *P* contains a <comment>.

5) Case:

a) If *E1* is followed by an <interval qualifier> *IQ*, then the data type of *E1* is assumed to be INTERVAL *IQ*.

b) In OCTET_LENGTH(*E1*), CHARACTER_LENGTH(*E1*), and CHARACTER_LENGTH(*E1*), the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

c) In POSITION(*X1* IN *X2*), and SUBSTRING(*X1* FROM *X3* FOR *X4*), if *X1* (*X2*) meets the criteria for *E1*, *E2*, *E3*, and *E4*, then the data type of *X1* (*X2*) is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING. If *X3* (*X4*) meets the criteria for *E1*, *E2*, *E3*, and *E4*, then the data type of *X3* (*X4*) is assumed to be NUMERIC(*P*,0), where *P* is the implementation-defined maximum value of <precision> for NUMERIC.

d) In a <value expression> of the form "*X1* <concatenation operator> *X2*", if *X1* (*X2*) meets the criteria for *E1*, *E2*, *E3*, and *E4*, then the data type of *X1* (*X2*) is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

e) In BIT_LENGTH(*E1*), the data type of *E1* is assumed to be BIT VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for BIT VARYING.

f) In a <value expression> of the form "*E1* + <datetime term>", "<datetime term> + *E1*" or "<datetime term> – *E1*", the data type of *E1* is assumed to be

Case:

i) If the <datetime term> is a date data type, then the data type of *E1* is assumed to be INTERVAL YEAR(*P*) TO MONTH, where *P* is the implementation-defined maximum <interval leading field precision>.

ii) Otherwise, the data type of *E1* is assumed to be INTERVAL DAY(*P*) TO SECOND(*F*), where *P* and *F* are the implementation-defined maximum <interval leading field precision> and maximum <interval fractional seconds precision>, respectively.

g) In a <value expression> of the form "<interval term> * *E1*" or "<interval term> / *E1*", the data type of *E1* is assumed to be NUMERIC(*P*,0), where *P* is the implementation-defined maximum value of <precision> for NUMERIC.

h) In all other <value expression>s of the form "*E1*+ *F*", "*E1*– *F*", "*E1** *F*", "*E1*/ *F*", "*F* + *E1*", "*F* – *E1*", "*F* * *E1*", or "*F* / *E1*", the data type of *E1* is assumed to be the data type of *F*.

i) In a <value expression> of the form "CAST (*E1* AS <domain name>)", "CAST (*E1* AS <data type>)", the data type of *E1* is the <data type> of the specified domain or the explicitly-specified <data type>.

j) If one or more operands of COALESCE are *E1*, then the data type of *E1* is assumed to be the data type of the first operand.

k) If one or more <when operand>s in a <case specification> are *E1*, then the data type of *E1* is assumed to be the data type of the first <when operand>.

l) If one operand of NULLIF is *E1*, then the data type of *E1* is assumed to be the data type of the other operand.

m) In the first and second operands of a <comparison predicate> or <between predicate>, or the first and third operands of a <between predicate>, if the *i*-th value of one operand is *E1*, then the data type of E1 is the data type of the *i*-th value of the other operand.

n) In the first and second operands of an <overlaps predicate>, if either of the first <row value constructor element>s is *E1*, then the data type of *E1* is the data type of the first <row value constructor element> of the other operand. If both of the first <row value constructor element>s are *E1*, then the data type of each *E1* is assumed to be TIMESTAMP WITH TIME ZONE.

o) In a <table value constructor> in which the *i*-th <value expression> of some <row value constructor> is *E1* that contains a <row value constructor> whose *i*-th <value expression> is not *E1*, the data type of *E1* is the data type of the *i*-th <value expression> of the first <row value constructor> whose *i*-th <value expression> is not *E1*.

p) In a <table value constructor> in which the *i*-th <value expression> in each <row value constructor> is *E1* that is the <query expression> simply contained in an <insert statement>, the data type of *E1* is the data type of the corresponding column of the implicit or explicit <insert column list> contained in the <insert statement>.

q) In an <in predicate> that specifies a <table subquery>, the data types of <value expression>s *E1* in the <row value constructor> are assumed to be the same as the data types of the respective columns of the <table subquery>.

r) In an <in predicate> that specifies an <in value list>, if the <row value constructor> is not *E1*, then let *D* be its data type. Otherwise, let *D* be the data type of the first <value specification> of the <in value list>. The data type of any *E1* in the <in predicate> is assumed to be *D*.

s) If *E1* appears for <match value>, <pattern>, or <escape character> in <like predicate>, then the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

t) If any value in the <row value constructor> of a <quantified comparison predicate> or <match predicate> is *E1*, then the data type of *E1* is assumed to be the same as the data type of the respective column of the <table subquery>.

u) If <value specification> in <set catalog statement>, <set schema statement>, <set names statement>, or <set session authorization identifier statement> is *E1*, then the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

v) If <simple value specification> in <set local time zone statement> is *E1*, then the data type of *E1* is assumed to be CHARACTER VARYING(*L*), where *L* is the implementation-defined maximum value of <length> for CHARACTER VARYING.

w) If a <value expression> in a <set clause> is *E1*, then the data type of *E1* is assumed to be the same data type as the corresponding <object column>.

6) If the value of the <SQL statement name> identifies an existing prepared statement, then an implicit

DEALLOCATE PREPARE *SSN*

is executed, where *SSN* is the value of the <SQL statement name>.

7) *P* is prepared for execution.

8) If <extended statement name> is specified for the <SQL statement name>, then let *S* be the character string that is the value of the <simple target specification> and let *V* be the character string that is the result of

TRIM (BOTH ' ' FROM *S*)

If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid SQL statement identifier*.

9) Case:

a) If <extended statement name> is specified for the <SQL statement name>, then the value of the <extended statement name> is associated with the prepared statement. This value and explicit or implied <scope option> shall be specified for each <execute statement> or <allocate cursor statement> that is to be associated with this prepared statement.

b) If <statement name> is specified for the <SQL statement name>, then:

i) If *P* is a <cursor specification> and <statement name> is associated with a cursor *C* through a <dynamic declare cursor>, then an association is made between *C* and *P*. The association is preserved until the prepared statement is destroyed.

ii) If *P* is not a <cursor specification> and <statement name> is associated with a cursor *C* through a <dynamic declare cursor>, then an exception condition is raised: *dynamic SQL error— prepared statement is not a cursor specification*.

iii) Otherwise, the same <statement name> shall be specified for each <execute statement> that is to be associated with this prepared statement.

10) The validity of an <extended statement name> value or a <statement name> in an SQL-transaction different from the one in which the statement was prepared is implementation-dependent.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.7 <deallocate prepared statement>

Function

Deallocate SQL-statements that have been prepared with a <prepare statement>.

Format

```
<deallocate prepared statement> ::=
       DEALLOCATE PREPARE <SQL statement name>
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the <module> that contains the <deallocate prepared statement> shall also contain a <prepare statement> that specifies the same <statement name>.

Access Rules

None.

General Rules

1) If the <SQL statement name> does not identify a statement prepared in the scope of the <SQL statement name>, then an exception condition is raised: *invalid SQL statement name*.

2) If the value of <SQL statement name> identifies an existing prepared statement that is the <cursor specification> of an open cursor, then an exception condition is raised: *invalid cursor state*.

3) The prepared statement identified by the <SQL statement name> is destroyed. Any cursor that was allocated with an <allocate cursor statement> that is associated with the prepared statement identified by the <SQL statement name> is destroyed. If the value of the <SQL statement name> identifies an existing prepared statement that is a <cursor specification>, then any prepared statements that reference that cursor are destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <deallocate prepared statement>.

2) The following restrictions apply for Entry SQL in addition to Intermediate SQL restrictions:

   None.

17.8 <describe statement>

Function

Obtain information about the <select list> columns or <dynamic parameter specification>s contained in a prepared statement.

Format

```
<describe statement> ::=
         <describe input statement>
       | <describe output statement>

<describe input statement> ::=
       DESCRIBE INPUT <SQL statement name> <using descriptor>

<describe output statement> ::=
       DESCRIBE [ OUTPUT ] <SQL statement name> <using descriptor>
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the containing <module> shall contain a <prepare statement> whose <statement name> is the same as the <statement name> of the <describe statement>.

Access Rules

None.

General Rules

1) When the <describe statement> is executed, if the value of the <SQL statement name> does not identify a statement prepared in the scope of the <SQL statement name>, then an exception condition is raised: *invalid SQL statement name*.

2) The descriptor area is set with information as described in Subclause 17.9, "<using clause>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <describe input statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.9 <using clause>

Function

Describe the input/output variables for an <SQL dynamic statement>.

Format

```
<using clause> ::=
        <using arguments>
      | <using descriptor>

<using arguments> ::=
      { USING | INTO } <argument> [ { <comma> <argument> }... ]

<argument> ::= <target specification>

<using descriptor> ::=
      { USING | INTO } SQL DESCRIPTOR <descriptor name>
```

Syntax Rules

1) The keyword INTO shall appear in <using clause> only if the <using clause> is contained in a <dynamic fetch statement> or a <result using clause>.

Access Rules

None.

General Rules

1) The <identifier>s used to reference components of SQL descriptor areas are as shown in Table 17, "Data types of <key word>s used in SQL item descriptor areas".

2) If a <descriptor name> is specified in a <using clause>, then an SQL system descriptor area shall have been allocated and not yet deallocated whose name is the value of the <descriptor name>'s <simple value specification> and whose scope is that specified by the <scope option>. Otherwise, an exception condition is raised: *invalid SQL descriptor name*.

3) When a <describe output statement> is executed, a representation of the column descriptors of the <select list> columns for the prepared statement is stored in the specified SQL descriptor area as follows:

   a) Let *N* be the <occurrences> specified when the <descriptor name> was allocated.

   b) If the prepared statement that is being described is a <dynamic select statement> or a <dynamic single row select statement>, then let *T* be the table defined by the prepared statement and let *D* be the degree of *T*. Otherwise, let *D* be 0.

   c) COUNT is set to *D*.

   d) If *D* is greater than *N*, then a completion condition is raised: *warning—insufficient item descriptor areas*.

e) If *D* is 0 or *D* is greater than *N*, then no item descriptor areas are set. Otherwise, the first *D* item descriptor areas are set so that the *i*-th item descriptor area contains the descriptor of the *i*-th column of *T*. The descriptor of a column consists of values for TYPE, NULLABLE, NAME, UNNAMED, and other fields depending on the value of TYPE as described below. The DATA and INDICATOR fields are not relevant in this case. Those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values.

i) TYPE is set to a code, as shown in Table 18, "Codes used for SQL data types in Dynamic SQL", indicating the data type of the column.

ii) NULLABLE is set to 1 if the resulting column is possibly nullable and 0 otherwise.

iii) If the column name is implementation-dependent, then NAME is set to the implementation-dependent name of the column, and UNNAMED is set to 1. Otherwise, NAME is set to the <derived column> name for the column and UNNAMED is set to 0.

iv) Case:

1) If TYPE indicates a <character string type>, then: LENGTH is set to the length or maximum length in characters of the character string; OCTET_LENGTH is set to the maximum possible length in octets of the character string; CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set; and COLLATION_CATALOG, COLLATION_SCHEMA and COLLATION_NAME are set to the <collation name> of the character string's collation. If the subject <language clause> specifies C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string.

2) If TYPE indicates a <bit string type>, then LENGTH is set to the length or maximum length in bits of the bit string and OCTET_LENGTH is set to the maximum possible length in octets of the bit string.

3) If TYPE indicates an <exact numeric type>, then PRECISION and SCALE are set to the precision and scale of the exact numeric.

4) If TYPE indicates an <approximate numeric type>, then PRECISION is set to the precision of the approximate numeric.

5) If TYPE indicates a <datetime type>, then LENGTH is set to the length in positions of the datetime type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 19, "Codes associated with datetime data types in Dynamic SQL", to indicate the specific datetime data type, and PRECISION is set to the <time precision> or <timestamp precision>, if either is applicable.

6) If TYPE indicates an <interval type>, then DATETIME_INTERVAL_CODE is set to a code as specified in Table 20, "Codes used for <interval qualifier>s in Dynamic SQL", to indicate the <interval qualifier> of the interval data type, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision>, and PRECISION is set to the <interval fractional seconds precision>, if applicable.

4) When a <describe input statement> is executed, a descriptor for the <dynamic parameter specification>s for the prepared statement is stored in the specified SQL descriptor area as follows:

   a) Let *N* be the <occurrences> specified when the <descriptor name> was allocated.

   b) Let *D* be the number of <dynamic parameter specification>s in the prepared statement.

   c) COUNT is set to *D*.

   d) If *D* is greater than *N*, then a completion condition is raised: *warning—insufficient item descriptor areas*.

   e) If *D* is 0 or *D* is greater than *N*, then no item descriptor areas are set. Otherwise, the first *D* item descriptor areas are set so that the *i*-th item descriptor area contains a descriptor of the *i*-th <dynamic parameter specification>. The descriptor of a <dynamic parameter specification> consists of values for TYPE, NULLABLE, NAME, UNNAMED, and other fields depending on the value of TYPE as described below. The DATA and INDICATOR fields are not relevant in this case. Those fields and fields that are not applicable for a particular value of TYPE are set to implementation-dependent values.

      i) TYPE is set to a code, as shown in Table 18, "Codes used for SQL data types in Dynamic SQL", indicating the data type of the <dynamic parameter specification>.

      ii) NULLABLE is set to 1.
      Note: This indicates that the <dynamic parameter specification> can have the null value.

      iii) UNNAMED is set to 1 and NAME is set to an implementation-dependent value.

      iv) Case:

         1) If TYPE indicates a <character string type>, then: LENGTH is set to the length or maximum length in characters of the character string; OCTET_LENGTH is set to the maximum possible length in octets of the character string; CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA and CHARACTER_SET_NAME are set to the <character set name> of the character string's character set; and COLLATION_CATALOG, COLLATION_SCHEMA and COLLATION_NAME are set to the <collation name> of the character string's collation. If the subject <language clause> specifies C, then the lengths specified in LENGTH and OCTET_LENGTH do not include the implementation-defined null character that terminates a C character string.

         2) If TYPE indicates a <bit string type>, then LENGTH is set to the length or maximum length in bits of the bit string and OCTET_LENGTH is set to the maximum possible length in octets of the bit string.

         3) If TYPE indicates an <exact numeric type>, then PRECISION and SCALE are set to the precision and scale of the exact numeric.

         4) If TYPE indicates an <approximate numeric type>, then PRECISION is set to the precision of the approximate numeric.

         5) If TYPE indicates a <datetime type>, then LENGTH is set to the length in positions of the datetime type, DATETIME_INTERVAL_CODE is set to a code as specified in Table 19, "Codes associated with datetime data types in Dynamic SQL", to indicate

the specific datetime data type and PRECISION is set to the <time precision> or <timestamp precision>, if either is applicable.

6) If TYPE indicates an <interval type>, then DATETIME_INTERVAL_CODE is set to a code as specified in Table 20, "Codes used for <interval qualifier>s in Dynamic SQL", to indicate the <interval qualifier> of the interval data type, DATETIME_INTERVAL_PRECISION is set to the <interval leading field precision> and PRECISION is set to the <interval fractional seconds precision>, if applicable.

5) When a <using clause> is used in a <dynamic open statement> or as the <parameter using clause> in an <execute statement>, the <using clause> describes the <dynamic parameter specification> values for the <dynamic open statement> or the <execute statement>, respectively. Let *PS* be the prepared <dynamic select statement> referenced by the <dynamic open statement> or the prepared statement referenced by the <execute statement>, respectively.

Let *D* be the number of <dynamic parameter specification>s in *PS*.

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) If the value of COUNT is greater than the number of <occurrences> specified when the <descriptor name> was allocated or is less than zero, then an exception condition is raised: *dynamic SQL error—invalid descriptor count*.

ii) If the value of COUNT is not *D*, then an exception condition is raised: *dynamic SQL error— using clause does not match dynamic parameter specifications*.

iii) If the first *D* item descriptor areas are not valid as specified in Subclause 17.1, "Description of SQL item descriptor areas", then an exception condition is raised: *dynamic SQL error— using clause does not match dynamic parameter specifications*.

iv) If the value of INDICATOR is not negative, and the value of DATA is not a valid value of the data type indicated by TYPE, then an exception condition is raised: *dynamic SQL error— using clause does not match dynamic parameter specifications*.

c) Let *TDT* be the effective data type of the *i*-th <dynamic parameter specification>, defined to be the type represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME that would be set by a <describe input statement> to reflect the descriptor of the *i*-th dynamic parameter of *PS*.

Note: See the General Rules of Subclause 17.8, "<describe statement>".

d) Case:

i) If <using descriptor> is specified, then let *SDT* be the effective data type of the *i*-th <dynamic parameter specification> value as represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME in the *i*-th item descriptor area. If the INDICATOR value of

the *i*-th item descriptor area is not negative, then let *SV* be the value represented by the value of DATA with data type *SDT*. Otherwise, let *SV* be the null value.

ii) If <using arguments> is specified, then let *SDT* and *SV* be the data type and value, respectively, of the *i*-th <argument>.

e) If the <cast specification>

CAST (*SV* AS *TDT*)

violates the Syntax Rules of Subclause 6.10, "<cast specification>", then an exception condition is raised: *dynamic SQL error—restricted data type attribute violation*.

f) If the <cast specification>

CAST (S*V* AS *TDT*)

violates the General Rules of Subclause 6.10, "<cast specification>", then an exception condition is raised in accordance with the General Rules of Subclause 6.10, "<cast specification>".

g) The <cast specification>

CAST (*SV* AS *TDT*)

is effectively performed and is the value of the *i*-th dynamic parameter.

6) When a <using clause> is used in a <dynamic fetch statement> or as the <result using clause> of an <execute statement>, it describes the <target specification>s for the <dynamic fetch statement> or <execute statement>, respectively. Let *PS* be the prepared <dynamic select statement> referenced by the <dynamic fetch statement> or the prepared <dynamic single row select statement> referenced by the <execute statement>, respectively.

Let *D* be the degree of the table specified by *PS*.

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 target specifications*.

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

i) If the value of COUNT is greater than the number of <occurrences> specified when the <descriptor name> was allocated or less than zero, then an exception condition is raised: *dynamic SQL error—invalid descriptor count*.

ii) If COUNT is not equal to *D*, then an exception condition is raised: *dynamic SQL error—using clause does not match target specifications*.

iii) If the first *D* item descriptor areas are not valid as specified in Subclause 17.1, "Description of SQL item descriptor areas", then an exception condition is raised: *dynamic SQL error— using clause does not match target specifications*.

7) When a <using clause> is used in a <dynamic fetch statement> or as the <result using clause> of an <execute statement>, the result is a set of <target specification> values corresponding to the <select list> columns for the retrieved row. If <using descriptor> is specified, then the SQL descriptor area is set. The value of the *i*-th <target specification> is represented in the *i*-th item descriptor area as follows:

a) Let *SDT* be the effective data type of the *i*-th <select list> column, defined to be the type represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_

INTERVAL_PRECISION, DATETIME_INTERVAL_CODE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME that would be set by a <describe output statement> to reflect the description of the *i*-th <select list> column. Let *SV* be the value of the <select list> column, with data type *SDT*.

Note: See the General Rules of Subclause 17.8, "<describe statement>".

b) Let *TDT* be the effective data type of the *i*-th <target specification> as represented by the values of TYPE, LENGTH, PRECISION, SCALE, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME, in the *i*-th item descriptor area.

c) If the <cast specification>

CAST (*SV* AS *TDT*)

violates the Syntax Rules of Subclause 6.10, "<cast specification>", then an exception condition is raised: *dynamic SQL error— restricted data type attribute violation*.

d) If the <cast specification>

CAST (*SV* AS *TDT*)

violates the General Rules of Subclause 6.10, "<cast specification>", then an exception condition is raised in accordance with the General Rules of Subclause 6.10, "<cast specification>".

e) The <cast specification>

CAST (*SV* AS *TDT*)

is effectively performed, and is the value *TV* of the *i*-th <target specification>.

f) If *TV* is the null value, then the value of INDICATOR is set to -1.

g) If *TV* is not the null value, then:

i) The value of INDICATOR is set to 0 and the value of DATA is set to *TV*.

ii) Case:

1) If TYPE indicates CHARACTER VARYING or BIT VARYING, then RETURNED_LENGTH is set to the length in characters or bits, respectively, of *TV*, and RETURNED_OCTET_LENGTH is set to the length in octets of *TV*.

2) If *SDT* is CHARACTER VARYING or BIT VARYING, then RETURNED_LENGTH is set to the length in characters or bits, respectively, of *SV*, and RETURNED_OCTET_LENGTH is set to the length in octets of *SV*.

Note: All other values of the SQL descriptor area are unchanged.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

a) A <descriptor name> shall be a <literal>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.10 <execute statement>

Function

Associate input parameters and output targets with a prepared statement and execute the statement.

Format

```
<execute statement> ::=
       EXECUTE <SQL statement name>
         [ <result using clause> ]
         [ <parameter using clause> ]

<result using clause> ::= <using clause>

<parameter using clause> ::= <using clause>
```

Syntax Rules

1) If <SQL statement name> is a <statement name>, then the containing <module> shall contain a <prepare statement> whose <statement name> is the same as the <statement name> of the <execute statement>.

2) If <result using clause> is specified, then the <result using clause> shall contain either a <using arguments> or <using descriptor> that contains the keyword INTO.

Access Rules

None.

General Rules

1) When the <execute statement> is executed, if the <SQL statement name> does not identify a statement *P* previously prepared in the scope of the <SQL statement name>, then an exception condition is raised: *invalid SQL statement name*.

2) If *P* is a <dynamic select statement> that does not conform to the Format and Syntax Rules of a <dynamic single row select statement>, then an exception condition is raised: *dynamic SQL error—cursor specification cannot be executed*.

3) If *P* contains the <table name> of a created or declared local temporary table and if the <execute statement> is not in the same <module> as the <prepare statement> that prepared the prepared statement, then an exception condition is raised: *syntax rule or access rule violation in dynamic SQL statement*.

4) If *P* contains <dynamic parameter specification>s and a <parameter using clause> is not specified, then an exception condition is raised: *dynamic SQL error— using clause required for dynamic parameters*.

5) If P is a <dynamic single row select statement> and a <result using clause> is not specified, then an exception condition is raised: *dynamic SQL error—using clause required for result fields*.

6) If a <parameter using clause> that is <using descriptor> is specified, then the General Rules specified in Subclause 17.9, "<using clause>", for a <parameter using clause> in an <execute statement> are applied.

7) If *P* is a <dynamic single row select statement>, then the General Rules specified in Subclause 17.9, "<using clause>", for a <result using clause> in an <execute statement> are applied.

8) *P* is executed.

   Case:

   a) If *P* is a <dynamic single row select statement>, then all General Rules in Subclause 13.5, "<select statement: single row>", apply to *P*, replacing "<query specification> *S*" with "the <query specification> contained in *P*". The <argument>s contained in the <result using clause> or the item descriptor areas of the SQL descriptor area referenced in the <result using clause>, if any, provide the <target specification>s corresponding to the <select list> of *P*.

   b) If the <preparable statement> is a <preparable dynamic delete statement: positioned>, then all General Rules in Subclause 17.19, "<preparable dynamic delete statement: positioned>", apply to the <preparable statement>.

   c) If the <preparable statement> is a <preparable dynamic update statement: positioned>, then all General Rules in Subclause 17.20, "<preparable dynamic update statement: positioned>", apply to the <preparable statement>.

   d) Otherwise, the results of the execution are the same as if the statement was contained in a <procedure> and executed; these are described in Subclause 12.3, "<procedure>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <result using clause>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.11 <execute immediate statement>

Function

Dynamically prepare and execute a preparable statement.

Format

```
<execute immediate statement> ::=
       EXECUTE IMMEDIATE <SQL statement variable>
```

Syntax Rules

1) The data type of <SQL statement variable> shall be character string.

Access Rules

None.

General Rules

1) Let *P* be the contents of the <SQL statement variable>.

2) If *P* is a <preparable dynamic delete statement: positioned> or a <preparable dynamic update statement: positioned>, then *P* refers to either a dynamic cursor with the same <cursor name> or to an extended dynamic cursor whose <extended cursor name> value is the same as the <cursor name>.

   Case:

   a) If both an extended dynamic cursor and a dynamic cursor with the same name as the <cursor name> exist, then an exception condition is raised: *ambiguous cursor name*.

   b) If there is neither an extended dynamic cursor nor a dynamic cursor with the name of <cursor name>, then an exception condition is raised: *invalid cursor name*.

3) If one or more of the following are true, then an exception condition is raised: *syntax error or access rule violation in dynamic SQL statement*.

   a) *P* does not conform to the Format, Syntax Rules, and Access Rules for a <preparable statement> or *P* is a <dynamic select statement> or a <dynamic single row select statement>.

   b) *P* contains a <comment>.

   c) *P* contains a <dynamic parameter specification>.

4) The <preparable statement> that is the value of the <SQL statement variable> is prepared and executed.

Case:

a) If the <preparable statement> is a <preparable dynamic delete statement: positioned>, then all General Rules in Subclause 17.19, "<preparable dynamic delete statement: positioned>", apply to the <preparable statement>.

b) If the <preparable statement> is a <preparable dynamic update statement: positioned>, then all General Rules in Subclause 17.20, "<preparable dynamic update statement: positioned>", apply to the <preparable statement>.

c) Otherwise, the results of the execution are the same as if the statement was contained in a <procedure> and executed; these are described in Subclause 12.3, "<procedure>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.12 <dynamic declare cursor>

Function

Declare a cursor to be associated with a <statement name>, which may in turn be associated with a <cursor specification>.

Format

```
<dynamic declare cursor> ::=
       DECLARE <cursor name> [ INSENSITIVE ] [ SCROLL ] CURSOR FOR <statement name>
```

Syntax Rules

1) The <cursor name> shall not be identical to the <cursor name> specified in any other <declare cursor> or <dynamic declare cursor> in the same <module>.

2) The containing <module> shall contain a <prepare statement> whose <statement name> is the same as the <statement name> of the <dynamic declare cursor>.

Access Rules

None.

General Rules

1) All General Rules of Subclause 13.1, "<declare cursor>" apply to <dynamic declare cursor>, replacing “<open statement>” with “<dynamic open statement>” and “<cursor specification>” with “prepared statement”.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <dynamic declare cursor> that specifies 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.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.13 <allocate cursor statement>

Function

Define a cursor based on a <prepare statement> for a <cursor specification>.

Format

```
<allocate cursor statement> ::=
       ALLOCATE <extended cursor name> [ INSENSITIVE ] [ SCROLL ] CURSOR
         FOR <extended statement name>
```

Syntax Rules

None.

Access Rules

None.

General Rules

1) When the <allocate cursor statement> is executed, if the value of the <extended statement name> does not identify a statement previously prepared in the scope of the <extended statement name>, then an exception condition is raised: *invalid SQL statement name.*

2) If the prepared statement associated with the <extended statement name> is not a <cursor specification>, then an exception condition is raised: *dynamic SQL error—prepared statement not a cursor specification*.

3) All General Rules of Subclause 13.1, "<declare cursor>" apply to <allocate cursor statement>, replacing "<open statement>" with "<dynamic open statement>" and "<cursor specification>" with "prepared statement".

4) Let *S* be the character string that is the value of the <simple value specification> immediately contained in <extended cursor name>. Let *V* be the character string that is the result of

   TRIM (BOTH ' ' FROM *S*)

   If *V* does not conform to the Format and Syntax Rules of an <identifier>, then an exception condition is raised: *invalid cursor name.*

5) If the value of the <extended cursor name> is identical to the value of the <extended cursor name> of any other cursor allocated in the scope of the <extended cursor name>, then an exception condition is raised: *invalid cursor name.*

6) An association is made between the value of the <extended cursor name> and the prepared statement in the scope of the <extended cursor name>. The association is preserved until the prepared statement is destroyed, at which time the cursor identified by <extended cursor name> is also destroyed.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall not contain any <allocate cursor statement>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

17.14 <dynamic open statement>

Function

Associate input parameters with a <cursor specification> and open the cursor.

Format

```
<dynamic open statement> ::=
       OPEN <dynamic cursor name> [ <using clause> ]
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

Access Rules

1) The Access Rules for the <query expression> simply contained in the prepared statement associated with the <dynamic cursor name> are applied.

General Rules

1) If <dynamic cursor name> is a <cursor name> and the <statement name> of the associated <dynamic declare cursor> is not associated with a prepared statement, then an exception condition is raised: *invalid SQL statement name*.

2) If <dynamic cursor name> is an <extended cursor name> whose value does not identify a cursor allocated in the scope of the <extended cursor name>, then an exception condition is raised: *invalid cursor name*.

3) If the prepared statement associated with the <dynamic cursor name> contains <dynamic parameter specification>s and a <using clause> is not specified, then an exception condition is raised: *dynamic SQL error—using clause required for dynamic parameters*.

4) The cursor specified by <dynamic cursor name> is updatable if and only if the associated <cursor specification> specified an updatable cursor.

   Note: *updatable cursor* is defined in Subclause 13.1, "<declare cursor>".

5) If a <using clause> is specified, then the General Rules specified in Subclause 17.9, "<using clause>", for <dynamic open statement> are applied.

6) All General Rules of Subclause 13.2, "<open statement>", apply to the <dynamic open statement>.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.15 <dynamic fetch statement>

Function

Fetch a row for a cursor declared with a <dynamic declare cursor>.

Format

```
<dynamic fetch statement> ::=
       FETCH [ [ <fetch orientation> ] FROM ] <dynamic cursor name> <using clause>
```

Syntax Rules

1) If <fetch orientation> is omitted, then NEXT is implicit.

2) The <using clause> shall specify INTO.

3) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

4) Let *CR* be the cursor identified by *DCN* and let *T* be the table defined by the <cursor specification> of *CR*.

5) If the implicit or explicit <fetch orientation> is not NEXT, then the <dynamic declare cursor> or <allocate cursor statement> associated with *CR* shall specify SCROLL.

6) The number of <target specification>s in <using arguments> or the number of item descriptor areas in the SQL descriptor area referenced by <using descriptor>, as appropriate, shall be the same as the degree of *T*. The *i*-th <target specification> in <using arguments> or the *i*-th item descriptor area of the SQL descriptor area, as appropriate, corresponds with the *i*-th column of *T*.

7) The Syntax Rules of Subclause 9.1, "Retrieval assignment", apply to each corresponding <target specification> in <using arguments> and each column of *T* as *TARGET* and *VALUE*, respectively.

Access Rules

None.

General Rules

1) The General Rules specified in Subclause 17.9, "<using clause>", for <dynamic fetch statement> are applied.

2) All General Rules of Subclause 13.3, "<fetch statement>", apply to the <dynamic fetch statement>, replacing “targets in the <fetch target list>” and “targets identified by the <fetch target list>” with “<target specification>s in the <using arguments> or item descriptor areas of the SQL descriptor area, as appropriate”.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.16 <dynamic close statement>

Function

Close a cursor.

Format

```
<dynamic close statement> ::=
       CLOSE <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

Access Rules

None.

General Rules

1) All General Rules of Subclause 13.4, "<close statement>", apply to the <dynamic close statement>.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.17 <dynamic delete statement: positioned>

Function

Delete a row of a table.

Format

```
<dynamic delete statement: positioned> ::=
       DELETE FROM <table name>
         WHERE CURRENT OF <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

3) *CR* shall be an updatable cursor.

   Note: *updatable cursor* is defined in Subclause 13.1, "<declare cursor>".

4) Let *T* be the table identified by the <table name>. 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>".

Access Rules

1) All Access Rules of Subclause 13.6, "<delete statement: positioned>", apply to the <dynamic delete statement: positioned>.

General Rules

1) All General Rules of Subclause 13.6, "<delete statement: positioned>", apply to the <dynamic delete statement: positioned>, replacing "<delete statement: positioned>" with "<dynamic delete statement: positioned>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.18 <dynamic update statement: positioned>

Function

Update a row of a table.

Format

```
<dynamic update statement: positioned> ::=
       UPDATE <table name>
         SET <set clause> [ { <comma> <set clause> }... ]
           WHERE CURRENT OF <dynamic cursor name>
```

Syntax Rules

1) If <dynamic cursor name> *DCN* is a <cursor name> *CN*, then the containing <module> shall contain a <dynamic declare cursor> whose <cursor name> is *CN*.

2) Let *CR* be the cursor identified by *DCN*.

3) *CR* shall be an updatable cursor.

   Note: *updatable cursor* is defined in Subclause 13.1, "<declare cursor>".

4) Let *T* be the table identified by the <table name>. 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>".

5) 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*.

6) 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>.

7) A <value expression> in a <set clause> shall not directly contain a <set function specification>.

8) The same <object column> shall not appear more than once in a <set clause list>.

9) If *CR* 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>.

10) The scope of the <table name> is the entire <update statement: positioned>.

11) For every <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.

Access Rules

1) All Access Rules of Subclause 13.9, "<update statement: positioned>", apply to the <dynamic update statement: positioned>.

General Rules

1) All General Rules of Subclause 13.9, "<update statement: positioned>", apply to the <dynamic update statement: positioned>, replacing “<cursor name>” with “<dynamic cursor name>” and “<update statement: positioned>” with “<dynamic update statement: positioned>”.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

17.19 <preparable dynamic delete statement: positioned>

Function

Delete a row of a table through a dynamic cursor.

Format

```
<preparable dynamic delete statement: positioned> ::=
       DELETE [ FROM <table name> ]
         WHERE CURRENT OF <cursor name>
```

Syntax Rules

1) If <table name> is not specified, then the name of the underlying table of the <cursor specification> identified by <cursor name> is implicit.

2) All Syntax Rules of Subclause 13.6, "<delete statement: positioned>", apply to the <preparable dynamic delete statement: positioned>, replacing "<declare cursor>" with "<dynamic declare cursor> or <allocate cursor statement>" and "<delete statement: positioned>" with "<preparable dynamic delete statement: positioned>".

Access Rules

1) All Access Rules of Subclause 13.6, "<delete statement: positioned>", apply to the <preparable dynamic delete statement: positioned>.

General Rules

1) All General Rules of Subclause 13.6, "<delete statement: positioned>", apply to the <preparable dynamic delete statement: positioned>, replacing "<delete statement: positioned>" with "<preparable dynamic delete statement: positioned>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <preparable dynamic delete statement: positioned>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

17.20 <preparable dynamic update statement: positioned>

Function

Update a row of a table through a dynamic cursor.

Format

```
<preparable dynamic update statement: positioned> ::=
       UPDATE [ <table name> ]
         SET <set clause list>
         WHERE CURRENT OF <cursor name>
```

Syntax Rules

1) If <table name> is not specified, then the name of the underlying table of the <cursor specification> identified by <cursor name> is implicit.

2) All Syntax Rules of Subclause 13.9, "<update statement: positioned>", apply to the <preparable dynamic update statement: positioned>, replacing "<declare cursor>" with "<dynamic declare cursor> or <allocate cursor statement>" and "<update statement: positioned>" with "<preparable dynamic update statement: positioned>".

Access Rules

1) All Access Rules of Subclause 13.9, "<update statement: positioned>", apply to the <preparable dynamic update statement: positioned>.

General Rules

1) All General Rules of Subclause 13.9, "<update statement: positioned>", apply to the <preparable dynamic update statement: positioned>, replacing "<update statement: positioned>" with "<preparable dynamic update statement: positioned>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) Conforming Intermediate SQL language shall contain no <preparable dynamic update statement: positioned>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

18 Diagnostics management

18.1 <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
       | DYNAMIC_FUNCTION
       | ROW_COUNT

<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
       | CATALOG_NAME
       | SCHEMA_NAME
       | TABLE_NAME
       | COLUMN_NAME
       | CURSOR_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 21, "<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 21—<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*)
DYNAMIC_FUNCTION	character varying (*L*)
ROW_COUNT	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*)
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*)
MESSAGE_TEXT	character varying (*L*)
MESSAGE_LENGTH	exact numeric with scale 0
MESSAGE_OCTET_LENGTH	exact numeric with scale 0
Where *L* is an implementation-defined integer not less than 128.	

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 22, "SQL-statement character codes for use in the diagnostics area" specifies the identifier of the SQL-statements.

d) The value of DYNAMIC_FUNCTION is the identification of the prepared statement executed. Table 22, "SQL-statement character codes for use in the diagnostics area", specifies the identifier of the SQL-statements.

Table 22—SQL-statement character codes for use in the diagnostics area

SQL-statement	Identifier
<allocate cursor statement>	ALLOCATE CURSOR
<allocate descriptor statement>	ALLOCATE DESCRIPTOR
<alter domain statement>	ALTER DOMAIN
<alter table statement>	ALTER TABLE
<assertion definition>	CREATE ASSERTION
<character set definition>	CREATE CHARACTER SET
<close statement>	CLOSE CURSOR
<collation definition>	CREATE COLLATION
<commit statement>	COMMIT WORK
<connect statement>	CONNECT
<deallocate descriptor statement>	DEALLOCATE DESCRIPTOR
<deallocate prepared statement>	DEALLOCATE PREPARE
<delete statement: positioned>	DELETE CURSOR
<delete statement: searched>	DELETE WHERE
<describe statement>	DESCRIBE
<direct select statement: multiple rows>	SELECT
<disconnect statement>	DISCONNECT
<domain definition>	CREATE DOMAIN
<drop assertion statement>	DROP ASSERTION
<drop character set statement>	DROP CHARACTER SET
<drop collation statement>	DROP COLLATION
<drop domain statement>	DROP DOMAIN

Table 22—SQL-statement character codes for use in the diagnostics area (Cont.)

SQL-statement	Identifier
<drop schema statement>	DROP SCHEMA
<drop table statement>	DROP TABLE
<drop translation statement>	DROP TRANSLATION
<drop view statement>	DROP VIEW
<dynamic close statement>	DYNAMIC CLOSE
<dynamic delete statement: positioned>	DYNAMIC DELETE CURSOR
<dynamic fetch statement>	DYNAMIC FETCH
<dynamic open statement>	DYNAMIC OPEN
<dynamic single row select statement>	SELECT
<dynamic update statement: positioned>	DYNAMIC UPDATE CURSOR
<execute immediate statement>	EXECUTE IMMEDIATE
<execute statement>	EXECUTE
<fetch statement>	FETCH
<get descriptor statement>	GET DESCRIPTOR
<get diagnostics statement>	GET DIAGNOSTICS
<grant statement>	GRANT
<insert statement>	INSERT
<open statement>	OPEN
<preparable dynamic delete statement: positioned>	DYNAMIC DELETE CURSOR
<preparable dynamic update statement: positioned>	DYNAMIC UPDATE CURSOR
<prepare statement>	PREPARE
<revoke statement>	REVOKE
<rollback statement>	ROLLBACK WORK
<schema definition>	CREATE SCHEMA
<select statement: single row>	SELECT
<set catalog statement>	SET CATALOG
<set connection statement>	SET CONNECTION
<set constraints mode statement>	SET CONSTRAINT
<set descriptor statement>	SET DESCRIPTOR
<set local time zone statement>	SET TIME ZONE
<set names statement>	SET NAMES
<set schema statement>	SET SCHEMA

Table 22—SQL-statement character codes for use in the diagnostics area (Cont.)

SQL-statement	Identifier
<set transaction statement>	SET TRANSACTION
<set session authorization identifier statement>	SET SESSION AUTHORIZATION
<table definition>	CREATE TABLE
<translation definition>	CREATE TRANSLATION
<update statement: positioned>	UPDATE CURSOR
<update statement: searched>	UPDATE WHERE
<view definition>	CREATE VIEW

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.

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 22.1, "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 22.1, "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 *triggered data change violation*, 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 <space>s.

      If TABLE_NAME identifies a declared local temporary table, then CATALOG_NAME is <space>s and SCHEMA_NAME is "MODULE".

g) If the value of RETURNED_SQLSTATE corresponds to *syntax error or access rule violation*, *syntax error or access rule violation in dynamic SQL statement*, or *syntax error or access rule violation in direct SQL statement*, then:

   i) 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 the 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 <space>s and SCHEMA_NAME contains "MODULE".

ii) If the syntax error or the 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 <space>s.

h) 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.

i) 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> and 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.

j) If the value of RETURNED_SQLSTATE does not correspond to *syntax error or access rule violation*, *syntax error or access rule violation in dynamic SQL statement*, or *syntax error or access rule violation in direct SQL statement*, 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>, 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>, 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>, 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.

k) 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.

l) The value of MESSAGE_LENGTH is the length in characters of the character string value in MESSAGE_TEXT.

m) The value of MESSAGE_OCTET_LENGTH is the length in octets of the character string value in MESSAGE_TEXT.

n) The values of CONNECTION_NAME and SERVER_NAME are

Case:

i) If COMMAND_FUNCTION or DYNAMIC_FUNCTION identifies an <SQL connection statement>, then the explicit or implicit <connection name> and the associated <SQL server specification>, respectively, referenced by the <SQL connection statement>.

ii) Otherwise, the explicit or implicit <connection name> and the associated <SQL-server name>, respectively, corresponding to the most-recently executed explicit or implicit <connect statement> or <set connection statement>.

o) The values of character string items where not otherwise specified by the preceding rules are set to 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 Intermediate SQL;

None.

2) The following restrictions apply for Entry SQL;

a) Conforming Entry SQL language shall not contain any <get diagnostics statement>.

19 Embedded SQL

19.1 <embedded SQL host program>

Function

Specify an <embedded SQL host program>.

Format

```
<embedded SQL host program> ::=
        <embedded SQL Ada program>
      | <embedded SQL C program>
      | <embedded SQL COBOL program>
      | <embedded SQL Fortran program>
      | <embedded SQL MUMPS program>
      | <embedded SQL Pascal program>
      | <embedded SQL PL/I program>

<embedded SQL statement> ::=
      <SQL prefix>
        <statement or declaration>
      [ <SQL terminator> ]

<statement or declaration> ::=
        <declare cursor>
      | <dynamic declare cursor>
      | <temporary table declaration>
      | <embedded exception declaration>
      | <SQL procedure statement>

<SQL prefix> ::=
        EXEC SQL
      | <ampersand>SQL<left paren>

<SQL terminator> ::=
        END-EXEC
      | <semicolon>
      | <right paren>

<embedded SQL declare section> ::=
        <embedded SQL begin declare>
          [ <embedded character set declaration> ]
          [ <host variable definition>... ]
        <embedded SQL end declare>
      | <embedded SQL MUMPS declare>

<embedded character set declaration> ::=
      SQL NAMES ARE <character set specification>

<embedded SQL begin declare> ::=
      <SQL prefix> BEGIN DECLARE SECTION [ <SQL terminator> ]

<embedded SQL end declare> ::=
      <SQL prefix> END DECLARE SECTION [ <SQL terminator> ]
```

19.1 <embedded SQL host program>

```
<embedded SQL MUMPS declare> ::=
       <SQL prefix>
         BEGIN DECLARE SECTION
           [ <embedded character set declaration> ]
           [ <host variable definition>... ]
         END DECLARE SECTION
       <SQL terminator>

<host variable definition> ::=
         <Ada variable definition>
       | <C variable definition>
       | <COBOL variable definition>
       | <Fortran variable definition>
       | <MUMPS variable definition>
       | <Pascal variable definition>
       | <PL/I variable definition>

<embedded variable name> ::=
       <colon><host identifier>

<host identifier> ::=
         <Ada host identifier>
       | <C host identifier>
       | <COBOL host identifier>
       | <Fortran host identifier>
       | <MUMPS host identifier>
       | <Pascal host identifier>
       | <PL/I host identifier>
```

Syntax Rules

1) An <embedded SQL host program> is a compilation unit that consists of programming language text and SQL text. The programming language text shall conform to the requirements of a specific standard programming language. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s, as defined in this American Standard.

   Note: *Compilation unit* is defined in Subclause 4.16, "Modules".

2) Case:

   a) An <embedded SQL statement> or <embedded SQL MUMPS declare> that is contained in an <embedded SQL MUMPS program> shall contain an <SQL prefix> that is "<ampersand>SQL<left paren>". There shall be no <separator> between the <ampersand> and "SQL" nor between "SQL" and the <left paren>.

   b) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> that is not contained in an <embedded SQL MUMPS program> shall contain an <SQL prefix> that is "EXEC SQL".

3) Case:

   a) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL COBOL program> shall contain an <SQL terminator> that is END-EXEC.

b) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL Fortran program> shall not contain an <SQL terminator>.

c) An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL Pascal program>, or <embedded SQL PL/I program> shall contain an <SQL terminator> that is a <semicolon>.

d) An <embedded SQL statement> or <embedded SQL MUMPS declare> that is contained in an <embedded SQL MUMPS program> shall contain an <SQL terminator> that is a <right paren>.

4) Case:

a) An <embedded SQL declare section> that is contained in an <embedded SQL MUMPS program> shall be an <embedded SQL MUMPS declare>.

b) An <embedded SQL declare section> that is not contained in an <embedded SQL MUMPS program> shall not be an <embedded SQL MUMPS declare>.

Note: There is no restriction on the number of <embedded SQL declare section>s that may be contained in an <embedded SQL host program>.

5) The <token>s comprising an <SQL prefix>, <embedded SQL begin declare>, or <embedded SQL end declare> shall be separated by <space> characters and shall be specified on one line. Otherwise, the rules for the continuation of lines and tokens from one line to the next and for the placement of host language comments are those of the programming language of the containing <embedded SQL host program>.

6) If an <embedded character set declaration> is not specified, then an <embedded character set declaration> that specifies an implementation-defined character set that contains at least every character that is in <SQL language character> is implicit.

7) A <temporary table declaration> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement or <declare cursor> that references the <table name> of the <temporary table declaration>.

8) A <declare cursor> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement that references the <cursor name> of the <declare cursor>.

9) A <dynamic declare cursor> that is contained in an <embedded SQL host program> shall precede in the text of that <embedded SQL host program> any SQL-statement that references the <cursor name> of the <dynamic declare cursor>.

10) Any <host identifier> that is contained in an <embedded SQL statement> in an <embedded SQL host program> shall be defined in exactly one <host variable definition> contained in that <embedded SQL host program>. In programming languages that support <host variable definition>s in subprograms, two <host variable definition>s with different, non-overlapping scope in the host language are to be regarded as defining different host variables, even if they specify the same variable name. That <host variable definition> shall appear in the text of the <embedded SQL host program> prior to any <embedded SQL statement> that references the <host identifier>. The <host variable definition> shall be such that a host language reference to the <host identifier> is valid at every <embedded SQL statement> that contains the <host identifier>.

11) A <host variable definition> defines the host language data type of the <host identifier>. For every such host language data type an equivalent SQL <data type> is specified in Subclause 19.3, "<embedded SQL Ada program>", Subclause 19.4, "<embedded SQL C program>", Subclause 19.5, "<embedded SQL COBOL program>", Subclause 19.6, "<embedded SQL Fortran program>", Subclause 19.7, "<embedded SQL MUMPS program>", Subclause 19.8, "<embedded SQL Pascal program>", and Subclause 19.9, "<embedded SQL PL/I program>".

12) If one or more <host variable definition>s that specify SQLSTATE or SQLCODE appear in an <embedded SQL host program>, then the <host variable definition>s shall be such that a host language reference to SQLSTATE or SQLCODE, respectively, is valid at every <embedded SQL statement>, including <embedded SQL statement>s that appear in any subprograms contained in that <embedded SQL host program>. No <embedded SQL statement> shall precede any of its applicable status code definitions in the text of the main program or subprograms that comprise the <embedded SQL host program>.

13) Given an <embedded SQL host program> *H*, there is an implied standard-conforming SQL <module> *M* and an implied standard-conforming host program *P* derived from *H*. The derivation of the implied program *P* and the implied <module> *M* of an <embedded SQL host program> *H* effectively precedes the processing of any host language program text manipulation commands such as inclusion or copying of text.

    Given an <embedded SQL host program> *H* with an implied <module> *M* and an implied program *P* defined as above:

    a) The implied <module> *M* of *H* shall be a standard-conforming SQL <module>.

    b) If *H* is an <embedded SQL Ada program>, an <embedded SQL C program>, an <embedded SQL COBOL program>, an <embedded SQL Fortran program>, an <embedded SQL MUMPS program>, an <embedded SQL Pascal program>, or an <embedded SQL PL/I program>, then the implied program *P* shall be a standard-conforming Ada program, a standard-conforming C program, a standard-conforming COBOL program, a standard-conforming Fortran program, a standard-conforming MUMPS program, a standard-conforming Pascal program, or standard-conforming PL/I program, respectively.

14) *M* is derived from *H* as follows:

    a) *M* contains a <module name clause> whose <module name> is either implementation-dependent or is omitted.

    b) *M* contains a <module character set specification> that is identical to the explicit or implicit <embedded character set declaration> with the keyword "SQL" removed.

    c) *M* contains a <language clause> that specifies either ADA, C, COBOL, FORTRAN, MUMPS, PASCAL, or PLI, where *H* is respectively an <embedded SQL Ada program>, an <embedded SQL C program>, an <embedded SQL COBOL program>, an <embedded SQL Fortran program>, an <embedded SQL MUMPS program>, an <embedded SQL Pascal program>, or an <embedded SQL PL/I program>.

    d) *M* contains a <module authorization clause> that specifies SCHEMA <schema name>, where the value of <schema name> is implementation-dependent.

    e) For every <declare cursor> *EC* contained in *H*, *M* contains one <declare cursor> *PC* and one <procedure> *PS* that contains an <open statement> that references *PC*. The <procedure name> of *PS* is implementation-dependent. *PC* is a copy of *EC* in which each distinct <embedded variable name> has been replaced with a distinct implementation-dependent <parameter name>. *PS* contains a <parameter declaration> for each <parameter name>

contained in *PC*. The <parameter declaration> that corresponds to a given <embedded variable name> *V* that is contained in *EC* specifies the <parameter name> with which *V* was replaced, and the SQL data type that corresponds to the host language data type of *V*. If *H* contains a <host variable definition> that specifies SQLSTATE or SQLCODE, then *PS* contains a <parameter declaration> that specifies SQLSTATE or SQLCODE, respectively. If *H* contains neither a <host variable definition> that specifies SQLSTATE nor a <host variable definition> that specifies SQLCODE, then *PS* contains a <parameter declaration> that specifies SQLCODE. The order of <parameter declaration>s in *PS* is implementation-dependent.

f) For every <dynamic declare cursor> *EC* in *H*, *M* contains one <dynamic declare cursor> *PC* that is a copy of *EC*.

g) *M* contains one <temporary table declaration> for each <temporary table declaration> contained in *H*. Each <temporary table declaration> of *M* is a copy of the corresponding <temporary table declaration> of *H*.

h) *M* contains a <procedure> for each <SQL procedure statement> contained in *H*. The <procedure> *PS* of *M* corresponding with an <SQL procedure statement> *ES* of *H* is defined as follows:

Case:

i) If *ES* is not an <open statement>, then:

1) The <procedure name> of *PS* is implementation-dependent.

2) The <SQL procedure statement> of *PS* is a copy of *ES* in which each distinct <embedded variable name> has been replaced with the same distinct implementation-dependent <parameter name>.

3) *PS* contains a <parameter declaration> for each distinct <parameter name> contained in the <SQL procedure statement> of *PS*.

4) The <parameter declaration> corresponding to a given <embedded variable name> *V* that is contained in *ES* specifies the <parameter name> with which *V* was replaced and the SQL <data type> that corresponds to the host language data type of *V*.

5) Whether one <procedure> of *M* can correspond to more than one <SQL procedure statement> of *H* is implementation-dependent.

6) If *H* contains a <host variable definition> that specifies SQLSTATE or SQLCODE, then *PS* contains a <parameter declaration> that specifies SQLSTATE or SQLCODE, respectively. If *H* contains neither a <host variable definition> that specifies SQLSTATE nor a <host variable definition> that specifies SQLCODE, then *PS* contains a <parameter declaration> that specifies SQLCODE.

7) The order of the <parameter declaration>s is implementation-dependent.

ii) If *ES* is an <open statement>, then:

1) Let *EC* be the <declare cursor> in *H* referenced by *ES*.

2) *PS* is the procedure in *M* that contains an <open statement> that references the <declare cursor> in M corresponding to *EC*.

15) *P* is derived from *H* as follows:

a) Each <embedded SQL begin declare>, <embedded SQL end declare>, and <embedded character set declaration> has been deleted. If the embedded host language is MUMPS, then each <embedded SQL MUMPS declare> has been deleted.

b) Each <host variable definition> in an <embedded SQL declare section> has been replaced by a valid data definition in the target host language according to the Syntax Rules specified in an <embedded SQL Ada program>, <embedded SQL C program>, <embedded SQL COBOL program>, <embedded SQL Fortran program>, <embedded SQL Pascal program>, or an <embedded SQL PL/I program> clause.

c) Each <embedded SQL statement> that contains a <declare cursor>, a <dynamic declare cursor>, or a <temporary table declaration> has been deleted, and every <embedded SQL statement> that contains an <embedded exception declaration> has been replaced with statements of the host language that will have the effect specified by the General Rules of Subclause 19.2, "<embedded exception declaration>".

d) Each <embedded SQL statement> that contains an <SQL procedure statement> has been replaced by host language statements that perform the following actions:

   i) A host language procedure or subroutine call of the <procedure> of the implied <module> *M* of *H* that corresponds with the <SQL procedure statement>.

      If the <SQL procedure statement> is not an <open statement>, then the arguments of the call include each distinct <host identifier> contained in the <SQL procedure statement> together with the SQLCODE <host identifier>, or the SQLSTATE <host identifier>, or both. If *H* contains neither a <host variable definition> that specifies SQLSTATE nor a <host variable definition> that specifies SQLCODE, then the arguments of the call include SQLCODE. If the <SQL procedure statement> is an <open statement>, then the arguments of the call include each distinct <host identifier> contained in the corresponding <declare cursor> of *H* together with the SQLCODE <host identifier>, or the SQLSTATE <host identifier>, or both. If *H* contains neither a <host variable definition> that specifies SQLSTATE nor a <host variable definition> that specifies SQLCODE, then the arguments of the call include SQLCODE.

      The order of the arguments in the call corresponds with the order of the corresponding <parameter declaration>s in the corresponding <procedure>.

      Note: In an <embedded SQL Fortran program>, the "SQLCODE" variable is abbreviated to "SQLCOD" and the "SQLSTATE" variable may be abbreviated to "SQLSTA". See the Syntax Rules of Subclause 19.6, "<embedded SQL Fortran program>".

   ii) Exception actions, as specified in Subclause 19.2, "<embedded exception declaration>".

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 American Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) The interpretation of an <embedded SQL host program> *H* is defined to be equivalent to the interpretation of the implied program *P* of *H* and the implied <module> *M* of *H*.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) An <embedded SQL declare section> shall not contain an <embedded character set declaration>.

   b) No two <host variable definition>s shall specify the same variable name.

19.2 <embedded exception declaration>

Function

Specify the action to be taken when an SQL-statement causes a specific class of condition to be raised.

Format

```
<embedded exception declaration> ::=
       WHENEVER <condition> <condition action>

<condition> ::=
       SQLERROR | NOT FOUND

<condition action> ::=
       CONTINUE | <go to>

<go to> ::=
       { GOTO | GO TO } <goto target>

<goto target> ::=
         <host label identifier>
       | <unsigned integer>
       | <host PL/I label variable>

<host label identifier> ::= !!See the Syntax Rules.

<host PL/I label variable> ::= !!See the Syntax Rules.
```

Syntax Rules

1) An <embedded exception declaration> contained in an <embedded SQL host program> applies to an <SQL procedure statement> contained in that <embedded SQL host program> if and only if the <SQL procedure statement> appears after the <embedded exception declaration> in the text sequence of the <embedded SQL host program> and no other <embedded exception declaration> that specifies the same <condition> appears between the <embedded exception declaration> and the <SQL procedure statement> in the text sequence of the <embedded SQL host program>.

2) If an <embedded exception declaration> specifies a <go to>, then the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to> shall be such that a host language GO TO statement specifying that <host label identifier>, <host PL/I label variable>, or <unsigned integer> is valid at every <SQL procedure statement> to which the <embedded exception declaration> applies.

   Note:

   — If an <embedded exception declaration> is contained in an <embedded SQL Ada program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a label_name in the containing <embedded SQL Ada program>.

   — If an <embedded exception declaration> is contained in an <embedded SQL C program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a label in the containing <embedded SQL C program>.

— If an <embedded exception declaration> is contained in an <embedded SQL COBOL program>, then the <goto target> of a <go to> should specify a <host label identifier> that is a section-name or an unqualified paragraph-name in the containing <embedded SQL COBOL program>.

— If an <embedded exception declaration> is contained in an <embedded SQL Fortran program>, then the <goto target> of a <go to> should be an <unsigned integer> that is the statement label of an executable statement that appears in the same program unit as the <go to>.

— If an <embedded exception declaration> is contained in an <embedded SQL MUMPS program>, then the <goto target> of a <go to> should be a gotoargument that is the statement label of an executable statement that appears in the same <embedded SQL MUMPS program>.

— If an <embedded exception declaration> is contained in an <embedded SQL Pascal program>, then the <goto target> of a <go to> should be an <unsigned integer> that is a label.

— If an <embedded exception declaration> is contained in an <embedded SQL PL/I program>, then the <goto target> of a <go to> should specify either a <host label identifier> or a <host PL/I label variable>.

Case:

- If <host label identifier> is specified, then the <host label identifier> should be a label constant in the containing <embedded SQL PL/I program>.
- If <host PL/I label variable> is specified, then the <host PL/I label variable> should be a PL/I label variable declared in the containing <embedded SQL PL/I program>.

Access Rules

None.

General Rules

1) Immediately after the execution of an <SQL procedure statement> in an <embedded SQL host program>:

Case:

a) If the value of the status variable(s) indicates the completion condition *no data* and the <embedded SQL host program> contains an <embedded exception declaration> that applies to the <SQL procedure statement> and whose <condition> is NOT FOUND and whose <condition action> is a <go to>, then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to>.

b) If the value of the status variable(s) indicates an exception condition and the <embedded SQL host program> contains an <embedded exception declaration> that applies to the <SQL procedure statement> and whose <condition> is SQLERROR and whose <condition action> is a <go to>, then a GO TO statement of the host language is performed, specifying the <host label identifier>, <host PL/I label variable>, or <unsigned integer> of the <go to>.

c) If the <embedded SQL host program> contains no <embedded exception declaration> that applies to the <SQL procedure statement>, or if it contains an <embedded exception declaration> that applies to the <SQL procedure statement> and that specifies CONTINUE, then no further action is performed for the <SQL procedure statement>.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

19.3 <embedded SQL Ada program>

Function

Specify an <embedded SQL Ada program>.

Format

```
<embedded SQL Ada program> ::= !! See the Syntax Rules.

<Ada variable definition> ::=
       <Ada host identifier> [ { <comma> <Ada host identifier> }... ] :
       <Ada type specification> [ <Ada initial value> ]

<Ada initial value> ::=
       <Ada assignment operator> <character representation>...

<Ada assignment operator> ::= <colon><equals operator>

<Ada host identifier> ::= !! See the Syntax Rules.

<Ada type specification> ::=
         <Ada qualified type specification>
       | <Ada unqualified type specification>

<Ada qualified type specification> ::=
         SQL_STANDARD.CHAR [ CHARACTER SET [ IS ] <character set specification> ]
             <left paren> 1 <double period> <length> <right paren>
       | SQL_STANDARD.BIT <left paren> 1 <double period> <length> <right paren>
       | SQL_STANDARD.SMALLINT
       | SQL_STANDARD.INT
       | SQL_STANDARD.REAL
       | SQL_STANDARD.DOUBLE_PRECISION
       | SQL_STANDARD.SQLCODE_TYPE
       | SQL_STANDARD.SQLSTATE_TYPE
       | SQL_STANDARD.INDICATOR_TYPE

<Ada unqualified type specification> ::=
         CHAR <left paren> 1 <double period> <length> <right paren>
       | BIT <left paren> 1 <double period> <length> <right paren>
       | SMALLINT
       | INT
       | REAL
       | DOUBLE_PRECISION
       | SQLCODE_TYPE
       | SQLSTATE_TYPE
       | INDICATOR_TYPE
```

Syntax Rules

1) An <embedded SQL Ada program> is a compilation unit that consists of Ada text and SQL text. The Ada text shall conform to the Ada standard ANSI/MIL-STD-1815A. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever an Ada statement may be specified. An <embedded SQL statement> may be prefixed by an Ada label.

3) An <Ada host identifier> is any valid Ada identifier. An <Ada host identifier> shall be contained in an <embedded SQL Ada program>.

4) An <Ada variable definition> defines one or more host variables.

5) An <Ada variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Ada program> (see the Syntax Rules of Subclause 19.1, "<embedded SQL host program>"):

   a) Any optional CHARACTER SET specification shall be removed from an <Ada qualified type specification>.

   b) The <length> specified in a CHAR declaration of any <Ada qualified type specification> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Ada host identifier> specified in the containing <Ada variable definition>.

   The modified <Ada variable definition> shall be a valid Ada object-declaration in the program derived from the <embedded SQL Ada program>.

6) An <Ada variable definition> shall be specified within the scope of Ada **with** and **use** clauses that specify the following:

   with SQL_STANDARD;

   use SQL_STANDARD;

   use SQL_STANDARD.CHARACTER_SET;

7) The <character representation> sequence in an <Ada initial value> specifies an initial value to be assigned to the Ada variable. It shall be a valid Ada specification of an initial value.

8) CHAR describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

9) BIT describes a bit string variable. The equivalent SQL data type is BIT with the same length.

10) INT and SMALLINT describe exact numeric variables. The equivalent SQL data types are INTEGER and SMALLINT, respectively.

11) REAL and DOUBLE_PRECISION describe approximate numeric variables. The equivalent SQL data types are REAL and DOUBLE PRECISION, respectively.

12) SQLCODE_TYPE describes an exact numeric variable whose precision is the implementation-defined precision defined for the SQLCODE parameter. SQLSTATE_TYPE describes a character string variable whose length is the length of the SQLSTATE parameter (five characters).

13) INDICATOR_TYPE describes an exact numeric variable whose specific data type is any <exact numeric type> with a scale of 0.

14) If an <embedded SQL Ada program> contains neither an <Ada variable definition> that specifies SQLSTATE and that is defined with an <Ada type specification> that specifies SQL_STANDARD.CHAR or CHAR with <length> 5, nor an <Ada variable definition> that specifies SQLCODE and that is defined with an <Ada type specification> that specifies SQL_

STANDARD.INTEGER or INTEGER, then it is assumed that the <embedded SQL Ada program> contains a variable named SQLCODE defined with a data type of INTEGER.

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 American Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 19.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) An <Ada variable definition> shall not specify a bit string variable.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) An <Ada qualified type specification> shall not contain a <character set specification>.

19.4 <embedded SQL C program>

Function

Specify an <embedded SQL C program>.

Format

```
<embedded SQL C program> ::= !! See the Syntax Rules.

<C variable definition> ::=
        [ <C storage class> ]
        [ <C class modifier> ]
        <C variable specification>
      <semicolon>

<C variable specification> ::=
        <C numeric variable>
      | <C character variable>
      | <C derived variable>

<C storage class> ::=
        auto
      | extern
      | static

<C class modifier> ::= const | volatile

<C numeric variable> ::=
      { long | short | float | double }
        <C host identifier> [ <C initial value> ]
              [ { <comma> <C host identifier> [ <C initial value> ] }... ]

<C character variable> ::=
      char [ CHARACTER SET [ IS ] <character set specification> ]
        <C host identifier> <C array specification> [ <C initial value> ]
          [ { <comma> <C host identifier> <C array specification>
                    [ <C initial value> ] }... ]

<C array specification> ::=
      <left bracket> <length> <right bracket>

<C host identifier> ::= !! See the Syntax Rules.

<C derived variable> ::=
        <C VARCHAR variable>
      | <C bit variable>

<C VARCHAR variable> ::=
      VARCHAR [ CHARACTER SET [ IS ] <character set specification> ]
          <C host identifier> <C array specification> [ <C initial value> ]
            [ { <comma> <C host identifier> <C array specification>
                        [ <C initial value> ] }... ]

<C bit variable> ::=
      BIT <C host identifier> <C array specification> [ <C initial value> ]
        [ { <comma> <C host identifier> <C array specification>
                    [ <C initial value> ] }... ]
```

```
<C initial value> ::=
      <equals operator> <character representation>...
```

Syntax Rules

1) An <embedded SQL C program> is a compilation unit that consists of C text and SQL text. The C text shall conform to the C standard ANSI X3.159. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a C statement may be specified within a function block. If the C statement could include a label prefix, then the <embedded SQL statement> may be immediately preceded by a label prefix.

3) A <C host identifier> is any valid C variable identifier. A <C host identifier> shall be contained in an <embedded SQL C program>.

4) A <C variable definition> defines one or more host variables.

5) A <C variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL C program> (see the Syntax Rules of Subclause 19.1, "<embedded SQL host program>"):

   a) Any optional CHARACTER SET specification shall be removed from a <C VARCHAR variable> or a <C character variable>.

   b) The syntax "VARCHAR" shall be replaced by "char" in any <C VARCHAR variable>.

   c) The syntax "BIT" shall be replaced by "char" in any <C bit variable>.

   d) The <length> specified in a <C array specification> in any <C bit variable> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

   e) The <length> specified in a <C array specification> in any <C character variable> or in any <C VARCHAR variable> that contained a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <C host identifier> specified in the containing <C variable definition>.

   The modified <C variable definition> shall be a valid C data declaration in the program derived from the <embedded SQL C program>.

6) The <character representation> sequence contained in a <C initial value> specifies an initial value to be assigned to the C variable. It shall be a valid C specification of an initial value.

7) Except for array specifications for character strings and bit strings, a <C variable definition> shall specify a scalar type.

8) In a <C variable definition>, the words "VARCHAR", "CHARACTER", "SET", "IS", "BIT", and "VARYING" may be specified in any combination of upper case and lower case letters (see the Syntax Rules of Subclause 5.2, "<token> and <separator>").

9) In a <C character variable> or a <C VARCHAR variable>, if a <character set specification> is specified, then the equivalent SQL data type is CHARACTER or CHARACTER VARYING whose character repertoire is the same as the repertoire specified by the <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

10) Each <C host identifier> specified in a <C character variable> describes a fixed-length character string. The length is specified by the <length> of the <C array specification>. The value in the host variable is terminated by a null character and the position occupied by this null character is included in the length of the host variable. The equivalent SQL data type is CHARACTER whose length is one less than the <length> of the <C array specification> and whose value does not include the terminating null character. The <length> shall be greater than 1.

11) Each <C host identifier> specified in a <C VARCHAR variable> describes a variable-length character string. The maximum length is specified by the <length> of the <C array specification>. The value in the host variable is terminated by a null character and the position occupied by this null character is included in the maximum length of the host variable. The equivalent SQL data type is CHARACTER VARYING whose maximum length is 1 less than the <length> of the <C array specification> and whose value does not include the terminating null character. The <length> shall be greater than 1.

12) Each <C host identifier> specified in a <C bit variable> describes a fixed-length bit string. The value in the host variable has a BIT_LENGTH of <length>. Let *B* be the number of bits in a C char and let *L* be the <length> of the <C array specification>. The length of an equivalent C char variable is the smallest integer that is not less than the result of *L*/*B*. The equivalent SQL data type is BIT whose length is *L*.

13) "long" describes an exact numeric variable. The equivalent SQL data type is INTEGER.

14) "short" describes an exact numeric variable. The equivalent SQL data type is SMALLINT.

15) "float" describes an approximate numeric variable. The equivalent SQL data type is REAL.

16) "double" describes an approximate numeric variable. The equivalent SQL data type is DOUBLE PRECISION.

17) If an <embedded SQL C program> contains neither a <C variable definition> that specifies SQLSTATE and that is defined with a <C character variable> that specifies "char" with a <C array specification> that is 6, nor a <C variable definition> that specifies SQLCODE and that is defined with a <C numeric variable> that specifies "long", then it is assumed that the <embedded SQL C program> contains a variable named SQLCODE defined with a data type of "long".

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 American Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 19.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <C derived variable> shall not be a <C bit variable>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <C derived variable> shall not be a <C VARCHAR variable>.

   b) A <C variable definition> shall not contain a <character set specification>.

19.5 <embedded SQL COBOL program>

Function

Specify an <embedded SQL COBOL program>.

Format

```
<embedded SQL COBOL program> ::= !! See the Syntax Rules.

<COBOL variable definition> ::=
       {01|77} <COBOL host identifier> <COBOL type specification>
         [ <character representation>... ] <period>

<COBOL host identifier> ::= !! See the Syntax Rules.

<COBOL type specification> ::=
         <COBOL character type>
       | <COBOL bit type>
       | <COBOL numeric type>
       | <COBOL integer type>

<COBOL character type> ::=
       [ CHARACTER SET [ IS ] <character set specification> ]
       { PIC | PICTURE } [ IS ] { X [ <left paren> <length> <right paren> ] }...

<COBOL bit type> ::=
       { PIC | PICTURE } [ IS ] { B [ <left paren> <length> <right paren> ] }...

<COBOL numeric type> ::=
       { PIC | PICTURE } [ IS ]
         S <COBOL nines specification>
       [ USAGE [ IS ] ] DISPLAY SIGN LEADING SEPARATE

<COBOL nines specification> ::=
         <COBOL nines> [ V [ <COBOL nines> ] ]
       | V <COBOL nines>

<COBOL integer type> ::=
         <COBOL computational integer>
       | <COBOL binary integer>

<COBOL computational integer> ::=
       { PIC | PICTURE } [ IS ] S<COBOL nines>
         [ USAGE [ IS ] ] { COMP | COMPUTATIONAL }

<COBOL binary integer> ::=
       { PIC | PICTURE } [ IS ] S<COBOL nines>
         [ USAGE [ IS ] ] BINARY

<COBOL nines> ::= { 9 [ <left paren> <length> <right paren> ] }...
```

Syntax Rules

1) An <embedded SQL COBOL program> is a compilation unit that consists of COBOL text and SQL text. The COBOL text shall conform to the COBOL standard ANSI X3.23. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> in an <embedded SQL COBOL program> may be specified wherever a COBOL statement may be specified in the Procedure Division of the <embedded SQL COBOL program>. If the COBOL statement could be immediately preceded by a paragraph-name, then the <embedded SQL statement> may be immediately preceded by a paragraph-name.

3) A <COBOL host identifier> is any valid COBOL data-name. A <COBOL host identifier> shall be contained in an <embedded SQL COBOL program>.

4) A <COBOL variable definition> is a restricted form of COBOL data description entry that defines a host variable.

5) A <COBOL variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL COBOL program> (see the Syntax Rules of Subclause 19.1, "<embedded SQL host program>".

   a) Any optional CHARACTER SET specification shall be removed from a <COBOL character type>.

   b) The syntax "B" shall be replaced by "X" in any <COBOL bit type>.

   c) The <length> specified in any <COBOL bit type> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

   d) The <length> specified in any <COBOL character type> that contained a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <COBOL host identifier> specified in the containing <COBOL variable definition>.

   The modified <COBOL variable definition> shall be a valid data description entry in the Data Division of the program derived from the <embedded SQL COBOL program>.

6) The optional <character representation> sequence in a <COBOL variable definition> may specify a VALUE clause. Whether other clauses may be specified is implementation-defined. The <character representation> sequence shall be such that the <COBOL variable definition> is a valid COBOL data description entry.

7) A <COBOL character type> describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

8) A <COBOL bit type> describes a bit string variable. Let *B* be the number of bits in a COBOL character and let *L* be the <length> of the <COBOL bit type>. The length of an equivalent COBOL character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the <COBOL bit type>.

9) A <COBOL numeric type> describes an exact numeric variable. The equivalent SQL data type is NUMERIC of the same precision and scale.

10) A <COBOL computational integer> describes an exact numeric variable.

Note: This <COBOL type specification> is supported only for SQLCODE for compatibility with earlier versions of this American Standard. The SQLCODE status parameter is a deprecated feature that is supported for compatibility with earlier versions of this American Standard. See Annex D, "Deprecated features".

11) A <COBOL binary integer> describes an exact numeric variable. The equivalent SQL data type is SMALLINT or INTEGER.

12) If an <embedded SQL COBOL program> contains neither a <COBOL variable definition> that specifies SQLSTATE and that is defined with a <COBOL character type> that specifies

{01 | 77} SQLSTATE PICTURE X(5)

nor a <COBOL variable definition> that specifies SQLCODE and that is defined with a <COBOL type specification> that specifies

{01 | 77} SQLCODE PICTURE S9(*PC*) USAGE COMP

where *PC* is the implementation-defined precision specified for a COBOL SQLCODE parameter in Subclause 12.4, "Calls to a <procedure>", then it is assumed that the <embedded SQL COBOL program> contains a variable named SQLCODE defined with a data type of

{01 | 77} SQLCODE PICTURE S9(*PC*) USAGE COMP

where *PC* is the implementation-defined precision specified for a COBOL SQLCODE parameter in Subclause 12.4, "Calls to a <procedure>".

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 American Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 19.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <COBOL type specification> shall not be a <COBOL bit type>.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <COBOL character type> shall not contain a <character set specification>.

19.6 <embedded SQL Fortran program>

Function

Specify an <embedded SQL Fortran program>.

Format

```
<embedded SQL Fortran program> ::= !! See the Syntax Rules.

<Fortran variable definition> ::=
       <Fortran type specification>
       <Fortran host identifier> [ { <comma> <Fortran host identifier> }... ]

<Fortran host identifier> ::= !! See the Syntax Rules.

<Fortran type specification> ::=
         CHARACTER [ <asterisk> <length> ]
             [ CHARACTER SET [ IS ] <character set specification> ]
       | BIT [ <asterisk> <length> ]
       | INTEGER
       | REAL
       | DOUBLE PRECISION
```

Syntax Rules

1) An <embedded SQL Fortran program> is a compilation unit that consists of Fortran text and SQL text. The Fortran text shall conform to the Fortran standards ANSI X3.9 and ANSI X3.198. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever an executable Fortran statement may be specified. An <embedded SQL statement> that precedes any executable Fortran statement in the containing <embedded SQL Fortran program> shall not have a Fortran statement number. Otherwise, if the Fortran statement could have a statement number then the <embedded SQL statement> can have a statement number.

3) Blanks are significant in <embedded SQL statement>s. The rules for <separator>s in an <embedded SQL statement> are as specified in Subclause 5.2, "<token> and <separator>".

4) A <Fortran host identifier> is any valid Fortran variable name with all <space> characters removed. A <Fortran host identifier> shall be contained in an <embedded SQL Fortran program>.

5) A <Fortran variable definition> is a restricted form of Fortran type-statement that defines one or more host variables.

6) A <Fortran variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Fortran program> (see the Syntax Rules Subclause 19.1, "<embedded SQL host program>".

   a) Any optional CHARACTER SET specification shall be removed from the CHARACTER alternative in a <Fortran type specification>.

b) The <length> specified in the CHARACTER alternative of any <Fortran type specification> that contained a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Fortran host identifier> specified in the containing <Fortran variable definition>.

c) The syntax "BIT" shall be replaced by "CHARACTER" in any BIT alternative of a <Fortran type specification>.

d) The <length> specified in any BIT alternative of a <Fortran type specification> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

The modified <Fortran variable definition> shall be a valid Fortran type-statement in the program derived from the <embedded SQL Fortran program>.

7) CHARACTER describes a character string variable whose equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

8) BIT describes a bit string variable. Let *B* be the number of bits in a Fortran CHARACTER and let *L* be the <length> of the bit string variable. The length of an equivalent Fortran character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the bit string variable.

9) INTEGER describes an exact numeric variable. The equivalent SQL data type is INTEGER.

10) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

11) DOUBLE PRECISION describes an approximate numeric variable. The equivalent SQL data type is DOUBLE PRECISION.

12) If an <embedded SQL Fortran program> contains neither a <Fortran variable definition> that specifies SQLSTATE or SQLSTA and that is defined with a <Fortran type specification> that specifies CHARACTER with <length> 5, nor a <Fortran variable definition> that specifies SQLCOD and that is defined with a <Fortran type specification> that specifies INTEGER, then it is assumed that the <embedded SQL Fortran program> contains a variable named SQLCOD defined with a data type of INTEGER.

Note: SQLSTATE (which may be abbreviated "SQLSTA") is the preferred status parameter. The SQLCODE (SQLCOD) status parameter is a deprecated feature that is supported for compatibility with earlier versions of this American Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 19.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <Fortran type specification> shall not specify BIT.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <Fortran type specification> shall not contain a <character set specification>.

19.7 <embedded SQL MUMPS program>

Function

Specify an <embedded SQL MUMPS program>.

Format

```
<embedded SQL MUMPS program> ::= !! See the Syntax Rules.

<MUMPS variable definition> ::=
       { <MUMPS numeric variable> | <MUMPS character variable> } <semicolon>

<MUMPS character variable> ::=
       VARCHAR <MUMPS host identifier> <MUMPS length specification>
         [ { <comma> <MUMPS host identifier> <MUMPS length specification> }... ]

<MUMPS host identifier> ::= !! See the Syntax Rules.

<MUMPS length specification> ::=
       <left paren> <length> <right paren>

<MUMPS numeric variable> ::=
       <MUMPS type specification>
         <MUMPS host identifier> [ { <comma> <MUMPS host identifier> }... ]

<MUMPS type specification> ::=
         INT
       | DEC [ <left paren> <precision> [ <comma> <scale> ] <right paren> ]
       | REAL
```

Syntax Rules

1) An <embedded SQL MUMPS program> is a compilation unit that consists of MUMPS text and SQL text. The MUMPS text shall conform to the MUMPS standard ANSI/MDC X11.1. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) A <MUMPS host identifier> is any valid MUMPS variable name. A <MUMPS host identifier> shall be contained in an <embedded SQL MUMPS program>.

3) An <embedded SQL statement> may be specified wherever a MUMPS command may be specified.

4) A <MUMPS variable definition> defines one or more host variables.

5) The <MUMPS character variable> describes a variable-length character string. The equivalent SQL data type is CHARACTER VARYING whose maximum length is the <length> of the <MUMPS length specification> and whose character set is implementation-defined.

6) INT describes an exact numeric variable. The equivalent SQL data type is INTEGER.

7) DEC describes an exact numeric variable. The <scale> shall not be greater than the <precision>. The equivalent SQL data type is DECIMAL with the same <precision> and <scale>.

8) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

9) If an <embedded SQL MUMPS program> contains neither a <MUMPS variable definition> that specifies SQLSTATE and that is defined with a <MUMPS character variable > that specifies VARCHAR with a <MUMPS length specification> that specifies 5, nor a <MUMPS variable definition> that specifies SQLCODE and that is defined with a <MUMPS numeric variable> that specifies INT, then it is assumed that the <embedded SQL MUMPS program> contains a variable named SQLCODE defined with a data type of INT.

   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 American Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 19.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) Conforming Entry SQL language shall contain no <embedded SQL MUMPS program>.

19.8 <embedded SQL Pascal program>

Function

Specify an <embedded SQL Pascal program>.

Format

```
<embedded SQL Pascal program> ::= !! See the Syntax Rules.

<Pascal variable definition> ::=
       <Pascal host identifier> [ { <comma> <Pascal host identifier> }... ] <colon>
         <Pascal type specification> <semicolon>

<Pascal host identifier> ::= !! See the Syntax Rules.

<Pascal type specification> ::=
         PACKED ARRAY <left bracket> 1 <double period> <length> <right bracket>
           OF CHAR
              [ CHARACTER SET [ IS ] <character set specification> ]
       | PACKED ARRAY <left bracket> 1 <double period> <length> <right bracket>
           OF BIT
       | INTEGER
       | REAL
       | CHAR [ CHARACTER SET [ IS ] <character set specification> ]
       | BIT
```

Syntax Rules

1) An <embedded SQL Pascal program> is a compilation unit that consists of Pascal text and SQL text. The Pascal text shall conform to one of the Pascal standards ANSI/IEEE 770/X3.97 and ANSI/IEEE 770/X3.160. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a Pascal statement may be specified. An <embedded SQL statement> may be prefixed by a Pascal label.

3) A <Pascal host identifier> is a Pascal variable-identifier whose applied instance denotes a defining instance within an <embedded SQL begin declare> and an <embedded SQL end declare>.

4) A <Pascal variable definition> defines one or more <Pascal host identifier>s.

5) A <Pascal variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL Pascal program> (see the Syntax Rules of Subclause 19.1, "<embedded SQL host program>").

   a) Any optional CHARACTER SET specification shall be removed from the PACKED ARRAY OF CHAR or CHAR alternatives of a <Pascal type specification>.

   b) The <length> specified in the PACKED ARRAY OF CHAR alternative of any <Pascal type specification> that contained a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <Pascal host identifier> specified in the containing <Pascal variable definition>.

c) If any <Pascal type specification> specifies the syntax "CHAR" and contains a CHARACTER SET specification, then let *L* be a length equal to the length in octets of *PN* and *PN* be the <Pascal host identifier> specified in the containing <Pascal variable definition>. If *L* is greater than 1, then "CHAR" shall be replaced by "PACKED ARRAY [1..*L*] OF CHAR".

d) The syntax "BIT" shall be replaced by "CHAR" in any PACKED ARRAY OF BIT or BIT alternatives of a <Pascal type specification>.

e) The <length> specified in any PACKED ARRAY OF BIT alternative in a <Pascal type specification> shall be replaced by a length equal to the smallest integer not less than *L*/*B*, as defined in the Syntax Rules of this Subclause.

The modified <Pascal variable definition> shall be a valid Pascal variable-declaration in the program derived from the <embedded SQL Pascal program>.

6) CHAR specified without a CHARACTER SET specification is the ordinal-type-identifier of PASCAL. The equivalent SQL data type is CHARACTER with length 1.

7) BIT describes a single-bit variable. It is mapped to a Pascal CHAR ordinal-type-identifier whose most significant bit contains either 0 or 1 and whose least significant bits contain 0. The equivalent SQL data type is BIT with length 1.

8) PACKED ARRAY [1..<length>] OF CHAR describes a character string having 2 or more components of the simple type CHAR. The equivalent SQL data type is CHARACTER with the same length and character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

9) PACKED ARRAY [1..<length>] OF BIT describes a bit string variable. Let *B* be the number of bits in a Pascal CHAR and let *L* be the <length> of the bit string variable. The length of an equivalent Pascal character variable is the smallest integer not less than *L*/*B*. The equivalent SQL data type is BIT whose length is the <length> of the bit string variable.

10) INTEGER describes an exact numeric variable. The equivalent SQL data type is INTEGER.

11) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

12) If an <embedded SQL Pascal program> contains neither a <Pascal variable definition> that specifies SQLSTATE and that is defined with a <Pascal type specification> that specifies PACKED ARRAY [1..<length>] OF CHAR with <length> 5, nor a <Pascal variable definition> that specifies SQLCODE and that is defined with a <Pascal type specification> that specifies INTEGER, then it is assumed that the <embedded SQL Pascal program> contains a variable named SQLCODE defined with a data type of INTEGER.

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 American Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 19.1, "<embedded SQL host program>".

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <Pascal type specification> shall not specify BIT or PACKED ARRAY [1..<length>] OF BIT.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <Pascal type specification> shall not contain a <character set specification>.

19.9 <embedded SQL PL/I program>

Function

Specify an <embedded SQL PL/I program>.

Format

```
<embedded SQL PL/I program> ::= !! See the Syntax Rules.

<PL/I variable definition> ::=
       {DCL | DECLARE}
          {   <PL/I host identifier>
            | <left paren> <PL/I host identifier>
                    [ { <comma> <PL/I host identifier> }... ] <right paren> }
       <PL/I type specification>
       [ <character representation>... ] <semicolon>

<PL/I host identifier> ::= !! See the Syntax Rules.

<PL/I type specification> ::=
         { CHAR | CHARACTER } [ VARYING ] <left paren><length><right paren>
             [ CHARACTER SET [ IS ] <character set specification> ]
       | BIT [ VARYING ] <left paren><length><right paren>
       | <PL/I type fixed decimal> <left paren> <precision>
             [ <comma> <scale> ] <right paren>
       | <PL/I type fixed binary> [ <left paren> <precision> <right paren> ]
       | <PL/I type float binary> <left paren> <precision> <right paren>

<PL/I type fixed decimal> ::=
         { DEC | DECIMAL } FIXED
       | FIXED { DEC | DECIMAL }

<PL/I type fixed binary> ::=
         { BIN | BINARY } FIXED
       | FIXED { BIN | BINARY }

<PL/I type float binary> ::=
         { BIN | BINARY } FLOAT
       | FLOAT { BIN | BINARY }
```

Syntax Rules

1) An <embedded SQL PL/I program> is a compilation unit that consists of PL/I text and SQL text. The PL/I text shall conform to the PL/I standard ANSI X3.53. The SQL text shall consist of one or more <embedded SQL statement>s and, optionally, one or more <embedded SQL declare section>s.

2) An <embedded SQL statement> may be specified wherever a PL/I statement may be specified within a procedure block. If the PL/I statement could include a label prefix, the <embedded SQL statement> may be immediately preceded by a label prefix.

3) A <PL/I host identifier> is any valid PL/I variable identifier. A <PL/I host identifier> shall be contained in an <embedded SQL PL/I program>.

4) A <PL/I variable definition> defines one or more host variables.

5) A <PL/I variable definition> shall be modified as follows before it is placed into the program derived from the <embedded SQL PL/I program> (see the Syntax Rules of Subclause 19.1, "<embedded SQL host program>").

   a) Any optional CHARACTER SET specification shall be removed from the CHARACTER or CHARACTER VARYING alternatives of a <PL/I type specification>.

   b) The <length> specified in the CHARACTER or CHARACTER VARYING alternatives of any <PL/I type specification> that contains a CHARACTER SET specification shall be replaced by a length equal to the length in octets of *PN*, where *PN* is the <PL/I host identifier> specified in the containing <PL/I variable definition>.

   The modified <PL/I variable definition> shall be a valid PL/I data declaration in the program derived from the <embedded SQL PL/I program>.

6) A <PL/I variable definition> shall specify a scalar variable, not an array or structure.

7) The optional <character representation> sequence in a <PL/I variable definition> may specify an INITIAL clause. Whether other clauses may be specified is implementation-defined. The <character representation> sequence shall be such that the <PL/I variable definition> is a valid PL/I DECLARE statement.

8) CHARACTER describes a character string variable whose equivalent SQL data type has the character set specified by <character set specification>. If <character set specification> is not specified, then an implementation-defined <character set specification> is implicit.

   Case:

   a) If VARYING is not specified, the length of the variable is fixed. The equivalent SQL data type is CHARACTER with the same length.

   b) If VARYING is specified, the variable is of variable length, with maximum size the value of <length>. The equivalent SQL data type is CHARACTER VARYING with the same maximum length.

9) BIT describes a bit string variable.

   Case:

   a) If VARYING is not specified, then the length of the variable is fixed. The equivalent SQL data type is BIT with the same length.

   b) If VARYING is specified, then the variable is of variable length with maximum size of the value of <length>. The equivalent SQL data type is BIT VARYING with the same maximum length.

10) FIXED DECIMAL describes an exact numeric variable. The <scale> shall not be greater than the <precision>. The equivalent SQL data type is DECIMAL with the same <precision> and <scale>.

11) FIXED BINARY describes an exact numeric variable. The equivalent SQL data type is SMALLINT or INTEGER.

12) FLOAT BINARY describes an approximate numeric variable. The equivalent SQL data type is FLOAT with the same <precision>.

13) If an <embedded SQL PL/I program> contains neither a <PL/I variable definition> that specifies SQLSTATE and that is defined with a <PL/I type specification> that specifies CHAR or CHARACTER with <length> 5 and does not specify VARYING, nor a <PL/I variable definition> that specifies SQLCODE and that is defined with a <PL/I type specification> that specifies FIXED BIN(*PP*) or FIXED BINARY(*PP*), where *PP* is the implementation-defined precision specified for a PL/I SQLCODE parameter in Subclause 12.4, "Calls to a <procedure>", then it is assumed that the <embedded SQL PL/I program> contains a variable named SQLCODE defined with a data type of FIXED BINARY(*PP*), where *PP* is the implementation-defined precision specified for a PL/I SQLCODE parameter in Subclause 12.4, "Calls to a <procedure>".

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 American Standard. See Annex D, "Deprecated features".

Access Rules

None.

General Rules

1) See Subclause 19.1, "<embedded SQL host program>"

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   a) A <PL/I type specification> shall not specify BIT or BIT VARYING.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <PL/I type specification> shall not specify CHARACTER VARYING.

   b) A <PL/I type specification> shall not contain a <character set specification>.

20 Direct invocation of SQL

20.1 <direct SQL statement>

Function

Specify direct execution of SQL.

Format

```
<direct SQL statement> ::=
       <directly executable statement> <semicolon>

<directly executable statement> ::=
         <direct SQL data statement>
       | <SQL schema statement>
       | <SQL transaction statement>
       | <SQL connection statement>
       | <SQL session statement>
       | <direct implementation-defined statement>

<direct SQL data statement> ::=
         <delete statement: searched>
       | <direct select statement: multiple rows>
       | <insert statement>
       | <update statement: searched>
       | <temporary table declaration>

<direct implementation-defined statement> ::= !!See the Syntax Rules
```

Syntax Rules

1) The <direct SQL data statement> shall not contain any <parameter specification>, <dynamic parameter specification>, or <variable specification>.

2) The <value specification> that represents the null value is implementation-defined.

3) The Format and Syntax Rules for <direct implementation-defined statement> are implementation-defined.

Access Rules

1) The Access Rules for <direct implementation-defined statement> are implementation-defined.

General Rules

1) The following <direct SQL statement>s are transaction-initiating <direct SQL statement>s:

   a) <direct SQL statement>s that are transaction-initiating <SQL procedure statement>s;

   b) <direct select statement: multiple rows>; and

c) <direct implementation-defined statement>s that are transaction-initiating.

2) After the last invocation of an SQL-statement by an SQL-agent in an SQL-session:

a) A <rollback statement> or a <commit statement> is effectively executed. If an unrecoverable error has occurred, or if the direct invocation of SQL 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. The determination of whether a direct invocation of SQL has terminated unexpectedly is implementation-dependent.

b) Let *D* be the <descriptor name> of any system descriptor area that is currently allocated within the current SQL-session. A <deallocate descriptor statement> that specifies

DEALLOCATE DESCRIPTOR *D*

is effectively executed.

c) All SQL-sessions associated with the SQL-agent are terminated.

3) Let *S* be the <direct SQL statement>.

4) The current <authorization identifier> for privilege determination for the execution of *S* is the SQL-session <authorization identifier>.

5) If *S* does not conform to the Format, Syntax Rules, and Access Rules for a <direct SQL statement>, then an exception condition is raised: *syntax error or access rule violation in direct SQL statement*.

6) When *S* is invoked by the SQL-agent:

Case:

a) If *S* is an <SQL connection statement>, then:

i) The diagnostics area is emptied.

ii) *S* is executed.

iii) If *S* successfully initiated or resumed an SQL-session, then subsequent invocations of a <direct SQL statement> by the SQL-agent are associated with that SQL-session until the SQL-agent terminates the SQL-session or makes it dormant.

b) 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 <procedure> or invocations of a <direct SQL statement> 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. If *S* is a <direct implementation-defined statement>, then it is implementation-defined whether or not *S* may be associated with an active SQL-transaction; if not, then an exception condition is raised: *invalid transaction state*.

iii) If no SQL-transaction is active for the SQL-agent, then

1) Case:

A) If *S* is a transaction-initiating <direct SQL statement>, then an SQL-transaction is initiated.

B) If *S* is a <direct implementation-defined statement>, then it is implementation-defined whether or not *S* initiates an SQL-transaction. If an implementation defines *S* to be transaction-initiating, then an SQL-transaction is initiated.

2) If *S* initiated an SQL-transaction, then:

A) Let *T* be the SQL-transaction initiated by *S*.

B) *T* is associated with this invocation and any subsequent invocations of <direct SQL statement>s or calls to a <procedure> by the SQL-agent until the SQL-agent terminates *T*.

C) Case:

I) If a <set transaction statement> has been executed since the termination of the last SQL-transaction in the SQL-session (or if there has been no previous SQL-transaction in the SQL-session and a <set transaction statement> has been executed), then the access mode, constraint mode, and isolation level of *T* are set as specified by the <set transaction statement>.

II) Otherwise, the access mode, constraint mode for all constraints, and isolation level for *T* are *read-write*, *immediate*, and SERIALIZABLE, respectively.

D) *T* is associated with the SQL-session.

iv) 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*.

v) The diagnostics area is emptied.

vi) *S* is executed.

7) If the execution of a <direct 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*.

8) Case:

   a) If *S* executed successfully, then either a completion condition is raised: *successful execution*, or a completion condition is raised: *warning*, or a completion condition is raised: *no data*, as determined by the General Rules in this and other Subclauses of this American Standard.

   b) If *S* did not execute successfully, then all changes made to SQL-data or schemas by the execution of *S* are canceled and an exception condition is raised as determined by the General Rules in this and other Subclauses of this American Standard.

   Note: The method of raising a condition is implementation-defined.

9) Diagnostics information resulting from the execution of *S* is placed into the diagnostics area as specified in Clause 18, "Diagnostics management".

   Note: The method of accessing the diagnostics information is implementation-defined, but does not alter the contents of the diagnostics area.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   a) A <direct SQL statement> shall not be an <SQL schema statement>.

20.2 <direct select statement: multiple rows>

Function

Specify a statement to retrieve multiple rows from a specified table.

Format

```
<direct select statement: multiple rows> ::=
       <query expression> [ <order by clause> ]
```

Syntax Rules

1) All Syntax Rules of Subclause 7.10, "<query expression>", apply to the <direct select statement: multiple rows>.

2) The <query expression> or <order by clause> of a <direct select statement: multiple rows> shall not contain any <value specification> other than a <literal>, CURRENT_USER, SESSION_USER, or SYSTEM_USER.

3) Let *T* be the table specified by the <query expression>.

4) If ORDER BY is specified, then each <sort specification> in the <order by clause> shall identify a column of *T*.

   Case:

   a) If a <sort specification> contains a <column name>, then *T* shall contain exactly one column with that <column name> and the <sort specification> identifies that column.

   b) If a <sort specification> contains an <unsigned integer>, then the <unsigned integer> shall be greater than 0 and not greater than the degree of *T*. The <sort specification> identifies the column of *T* with the ordinal position specified by the <unsigned integer>.

Access Rules

None.

General Rules

1) All General Rules of Subclause 7.10, "<query expression>", apply to the <direct select statement: multiple rows>.

2) Let *Q* be the result of the <query expression>.

3) If *Q* is empty, then a completion condition is raised: *no data*.

4) If an <order by clause> is not specified, then the ordering of the rows of *Q* 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 ASC is specified or implied 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 *X* and *Y* be distinct rows in the result table, and let XV_i and YV_i be the values of K_i in these rows, respectively. The relative position of rows *X* and *Y* in the result is determined by comparing XV_i and YV_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. XV_i is said to *precede* YV_i if the value of the <comparison predicate> "XV_i <comp op> YV_i" is true for the applicable <comp op>.

   c) In the result table, the relative position of row *X* is before row *Y* if and only if XV_n precedes YV_n for some *n* greater than 0 and less than the number of <sort specification>s and $XV_i = YV_i$ for all $i < n$. The relative order of two rows for which $XV_i = YV_i$ for all *i* is implementation-dependent.

6) If *Q* is not empty, then *Q* is returned. The method of returning *Q* is implementation-defined.

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

21 Information Schema and Definition Schema

21.1 Introduction

The views of the Information Schema are defined in terms of base tables. 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 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 American Standard. Constraints defined in this Clause are not actual SQL constraints.

<identifier>s are represented in the tables of the Information Schema by <string value expression>s 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). Comparison of <identifier>s is defined in Subclause 8.2, "<comparison predicate>". 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 before the <identifier> is stored in any table of the Information Schema).

21.2 Information Schema

21.2.1 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 Intermediate SQL:

   None.

2) 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.

21.2.2 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 ) )
```

Definition

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 Intermediate SQL:

   None.

2) 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.

21.2.3 INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY assertion

Function

The assertion INFORMATION_SCHEMA_CATALOG_NAME_CARDINALITY ensures 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.

21.2.4 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 Intermediate SQL:

   None.

2) 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.

21.2.5 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, DOMAIN_DEFAULT
       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 Intermediate SQL:

   None.

2) 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.

21.2.6 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, SCHEMA_NAME S) )
         WHERE
           SCHEMA_OWNER = CURRENT_USER
           AND CONSTRAINT_CATALOG
             = ( SELECT CATALOG_NAME FROM INFORMATION_SCHEMA_CATALOG_NAME )
```

Leveling Rules

1) The following restrictions apply for Intermediate SQL:

   None.

2) 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.

21.2.7 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 Intermediate SQL:

   None.

2) 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.

21.2.8 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 Intermediate SQL:

   None.

2) 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.

21.2.9 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,
         CASE WHEN 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 (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
       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 Intermediate SQL:

   None.

2) 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.

21.2.10 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 Intermediate SQL:

   None.

2) 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.

21.2.11 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 Intermediate SQL:

   None.

2) 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.

21.2.12 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 Intermediate SQL:

   None.

2) 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.

21.2.13 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 Intermediate SQL:

   None.

2) 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.

21.2.14 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 Intermediate SQL:

   None.

2) 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.

21.2.15 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 Intermediate SQL:

   None.

2) 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.

21.2.16 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 Intermediate SQL:

   None.

2) 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.

21.2.17 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 Intermediate SQL:

   None.

2) 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.

21.2.18 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 Intermediate SQL:

   None.

2) 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.

21.2.19 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
       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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall not reference the COLLATIONS view.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

21.2.20 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 Intermediate SQL:

   a) Conforming Intermediate SQL language shall not reference the TRANSLATIONS view.

2) The following restrictions apply for Entry SQL in addition to any Intermediate SQL restrictions:

   None.

21.2.21 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 Intermediate SQL:

   None.

2) 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.

21.2.22 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 Intermediate SQL:

   None.

2) 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.

21.2.23 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 Intermediate SQL:

   None.

2) 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.

21.2.24 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 Intermediate SQL:

   None.

2) 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.

21.2.25 COLUMN_DOMAIN_USAGE view

Function

Identify the columns defined in this catalog 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 Intermediate SQL:

   None.

2) 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.

21.2.26 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 Intermediate SQL:

   None.

2) 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.

21.2.27 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>.

21.2.28 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.

21.2.29 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).

21.3 Definition Schema

21.3.1 Introduction

The base tables are all defined in a <schema definition> for the schema named DEFINITION_SCHEMA. The table definitions are as complete as the definitional power of SQL allows. The table definitions are supplemented with assertions where appropriate; see Subclause 21.4, "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 4.13, "Clusters of catalogs").

21.3.2 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.

21.3.3 USERS base table

Function

The USERS table has one row for each <authorization identifier> referenced in the Information Schema of the catalog. 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
       )
```

Description

1) The values of USER_NAME are <authorization identifier>s that are known.

21.3.4 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.

21.3.5 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,

       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 ) 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
         OR
               DATA_TYPE IN ( 'INTEGER', 'SMALLINT', 'NUMERIC', 'DECIMAL' )
               AND ( CHARACTER_MAXIMUM_LENGTH, CHARACTER_OCTET_LENGTH,
                   COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME )
                       IS NULL
               AND NUMERIC_SCALE IS NOT NULL
               AND ( NUMERIC_SCALE <> 0 AND NUMERIC_PRECISION_RADIX = 10
                     OR
                     NUMERIC_SCALE = 0 AND NUMERIC_PRECISION_RADIX IN ( 2, 10 ) )
               AND DATETIME_PRECISION IS NULL
         OR
               DATA_TYPE IN ( 'DATE', 'TIME', 'TIMESTAMP',
                              'TIME WITH TIME ZONE', 'TIMESTAMP WITH TIME ZONE',
                              'INTERVAL' )
               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 NOT 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_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, and DATETIME_PRECISION 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, and the precision if it is a datetime or interval type.

21.3.6 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,

       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_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 default value. 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 11.5, "<default clause>".

21.3.7 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.

21.3.8 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 EXCEPTIONEXISTS
         ( 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.

21.3.9 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 reference>s that were contained in the <query expression> associated with the <view definition> are replaced by explicit <column reference>s 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 updatable.
NO	The <view definition> simply contains a <query expression> that is not updatable.

21.3.10 COLUMNS base table

Function

The COLUMNS table has one row for each 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 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_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 COLUMNS_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 )
             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 )
            ) )
       )
```

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) The value of ORDINAL_POSITION is the ordinal position of the column in the table.

6) The value of COLUMN_DEFAULT is null if the column being described has no default value or if its default value comes only from a domain. 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 11.5, "<default clause>".

7) The values of IS_NULLABLE have the following meanings:

YES	The columns is possibly nullable.
NO	The column is known not nullable.

21.3.11 VIEW_TABLE_USAGE base table

Function

The VIEW_TABLE_USAGE table has one row for each table referenced 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 view requires.

21.3.12 VIEW_COLUMN_USAGE base table

Function

The VIEW_COLUMN_USAGE table has one row for each column referenced by a view.

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 identifier, respectively, of a column that is explicitly or implicitly referenced in the <query expression> of the view being described.

21.3.13 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 NOT NULL,

       CONSTRAINT TABLE_CONSTRAINTS_PRIMARY_KEY
         PRIMARY KEY ( CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME ),

       CONSTRAINT TABLE_CONSTRAINTS_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 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.

21.3.14 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.

21.3.15 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', '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', '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 ( TABLE_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:

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.
NO ACTION	A <referential action> of NO ACTION was specified.

5) The values of DELETE_RULE have the following meanings for a referential constraint that has a <delete rule>:

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.
NO ACTION	A <referential action> of NO ACTION was specified.

21.3.16 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 reference>s that were contained in the <search condition> associated with a <check constraint definition> or an <assertion definition> are replaced by explicit <column reference>s in CHECK_CONSTRAINTS.

21.3.17 CHECK_TABLE_USAGE base table

Function

The CHECK_TABLE_USAGE table has one row for each table referenced by 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 that is referenced by the constraint being described.

21.3.18 CHECK_COLUMN_USAGE base table

Function

The CHECK_COLUMN_USAGE table has one row for each column referenced by 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, and qualified identifier, respectively, of a column that is referenced by the constraint being described.

21.3.19 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,

       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.

21.3.20 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_PRIVILEGES_TYPE_CHECK
           CHECK ( PRIVILEGE_TYPE IN
                     ( 'SELECT', 'INSERT', 'DELETE', 'UPDATE'
                       'REFERENCES' ) ),
       IS_GRANTABLE            INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT TABLE_PRIVILEGES_GRANTABLE_NOT_NULL NOT NULL
          CONSTRAINT TABLE_PRIVILEGES_GRANTABLE_CHECK
            CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT TABLE_PRIVILEGES_PRIMARY_KEY
         PRIMARY KEY ( GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
                       PRIVILEGE_TYPE ),

       CONSTRAINT TABLE_PRIVILEGES_FOREIGN_KEY_TABLES
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
           REFERENCES TABLES,

       CONSTRAINT TABLE_PRIVILEGES_GRANTOR_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,

       CONSTRAINT TABLE_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 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.

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.

21.3.21 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_PRIVILEGES_TYPE_CHECK
           CHECK ( PRIVILEGE_TYPE IN ( 'SELECT', 'INSERT', 'UPDATE', 'REFERENCES' ) ),
       IS_GRANTABLE     INFORMATION_SCHEMA.CHARACTER_DATA
         CONSTRAINT COLUMN_PRIVILEGES_IS_GRANTABLE_NOT_NULL NOT NULL
         CONSTRAINT COLUMN_PRIVILEGES_IS_GRANTABLE_CHECK
           CHECK ( IS_GRANTABLE IN ( 'YES', 'NO' ) ),

       CONSTRAINT COLUMN_PRIVILEGES_PRIMARY_KEY
         PRIMARY KEY
           ( GRANTOR, GRANTEE, TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME,
             PRIVILEGE_TYPE, COLUMN_NAME ),

       CONSTRAINT COLUMN_PRIVILEGES_FOREIGN_KEY_COLUMNS
         FOREIGN KEY ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME )
           REFERENCES COLUMNS,

       CONSTRAINT COLUMN_PRIVILEGES_GRANTOR_FOREIGN_KEY_USERS
         FOREIGN KEY ( GRANTOR )
           REFERENCES USERS,

       CONSTRAINT COLUMN_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 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.COLUMN_NAME.
INSERT	The user has INSERT privilege on the column identified by TABLE_CATALOG.TABLE_SCHEMA.TABLE_NAME.COLUMN_NAME.
UPDATE	The user has UPDATE privilege on the column identified by TABLE_CATALOG.TABLE_SCHEMA.TABLE_NAME.COLUMN_NAME.
REFERENCE	The user has REFERENCES privilege on the column identified by TABLE_CATALOG.TABLE_SCHEMA.TABLE_NAME.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.

21.3.22 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.SQL_IDENTIFIER
         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 ) ),

       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.

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.

21.3.23 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 ( CHARACTER_SET_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.

21.3.24 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' ) ),

       CONSTRAINT COLLATIONS_PRIMARY_KEY
         PRIMARY KEY ( COLLATION_CATALOG, COLLATION_SCHEMA, COLLATION_NAME ),

       CONSTRAINT COLLATIONS_FOREIGN_KEY_SCHEMATA
         FOREIGN KEY ( COLLATION_CATALOG, COLLATION_SCHEMA )
           REFERENCES SCHEMATA,

       CONSTRAINT COLLATIONS_CHECK_REFERENCES_CHARACTER_SETS
         CHECK ( COLLATION_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 defined.

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) A row always exists in this table for the collation 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.

21.3.25 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,

       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.

21.3.26 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 = 'DIRECT' AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUAGE IS NULL )
                         OR
                         ( SQL_LANGUAGE_BINDING_STYLE IN ( 'EMBEDDED', 'MODULE' )
                           AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUAGE IN
                             ( 'COBOL', 'FORTRAN', 'PASCAL', 'PLI' ) ) ) )
                     OR
                     ( SQL_LANGUAGE_YEAR = '1989' AND AND
                       SQL_LANGUAGE_CONFORMANCE IN ( '1', '2' ) AND
                       SQL_LANGUAGE_INTEGRITY IN ( 'NO', 'YES' ) AND
                       ( ( SQL_LANGUAGE_BINDING_STYLE = 'DIRECT' AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUAGE IS NULL )
                         OR
                         ( SQL_LANGUAGE_BINDING_STYLE IN ( 'EMBEDDED', '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  = 'DIRECT' AND
                           SQL_LANGUAGE_PROGRAMMING_LANGUGE IS NULL )
                         OR
                         ( SQL_LANGUAGE_BINDING  IN ( 'EMBEDDED', '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, direct invocation, or an embedded programming 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. If the value of SQL_LANGUAGE_BINDING_STYLE is 'EMBEDDED', then the binding style of <embedded SQL host program> is supported. If the value of SQL_LANGUAGE_BINDING_STYLE is 'DIRECT', then the binding style of <direct SQL statement> 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 'DIRECT', then SQL_LANGUAGE_PROGRAMMING_LANGUAGE is the null value. If the value of SQL_LANGUAGE_BINDING_STYLE is 'MODULE' or 'EMBEDDED', 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.

21.4 Assertions on the base tables

The following clauses specify assertions that apply to the base tables specified in Subclause 21.3, "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.

21.4.1 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.

21.4.2 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 ) )
```

21.4.3 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' ) ) )
```

22 Status codes

22.1 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 23, "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 American Standard or in any other 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 American 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 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*.

If a subclass value is not specified for a condition, then either subclass '000' or an implementation-defined subclass is returned.

An implementation-specified condition may duplicate a condition defined in this American Standard; however, if such a condition occurs as a result of executing a statement, then the corresponding implementation-defined SQLSTATE value shall not be returned in the SQLSTATE parameter but may be returned in the diagnostics area.

Table 23—SQLSTATE class and subclass values

Condition	Class	Subcondition	Subclass
ambiguous cursor name	3C	(*no subclass*)	000
cardinality violation	21	(*no subclass*)	000
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

Table 23—SQLSTATE class and subclass values (Cont.)

Condition	Class	Subcondition	Subclass
		transaction resolution unknown	007
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
		interval field overflow	015
		invalid character value for cast	018
		invalid datetime format	007
		invalid escape character	019
		invalid escape sequence	025
		invalid parameter value	023
		invalid time zone displacement value	009
		null value, no indicator parameter	002
		numeric value out of range	003
		string data, length mismatch	026
		string data, right truncation	001
		substring error	011
		trim error	027
		unterminated C string	024
dependent privilege descriptors still exist	2B	(*no subclass*)	000
dynamic SQL error	07	(*no subclass*)	000
		cursor specification cannot be executed	003
		invalid descriptor count	008
		invalid descriptor index	009
		prepared statement not a cursor specification	005
		restricted data type attribute violation	006
		using clause does not match dynamic parameter specifications	001
		using clause does not match target specifications	002
		using clause required for dynamic parameters	004

Table 23—SQLSTATE class and subclass values (Cont.)

Condition	Class	Subcondition	Subclass
		using clause required for result fields	007
feature not supported	0A	(*no subclass*)	000
		multiple server transactions	001
integrity constraint violation	23	(*no subclass*)	000
invalid authorization specification	28	(*no subclass*)	000
invalid catalog name	3D	(*no subclass*)	000
invalid character set name	2C	(*no subclass*)	000
invalid condition number	35	(*no subclass*)	000
invalid connection name	2E	(*no subclass*)	000
invalid cursor name	34	(*no subclass*)	000
invalid cursor state	24	(*no subclass*)	000
invalid schema name	3F	(*no subclass*)	000
invalid SQL descriptor name	33	(*no subclass*)	000
invalid SQL statement name	26	(*no subclass*)	000
invalid transaction state	25	(*no subclass*)	000
invalid transaction termination	2D	(*no subclass*)	000
no data	02	(*no subclass*)	000
Remote Database Access	HZ	(*See ISO/IEC 9579-2 for the definition of protocol subconditions and subclass code values*)	
successful completion	00	(*no subclass*)	000
syntax error or access rule violation	42	(*no subclass*)	000
syntax error or access rule violation in direct SQL statement	2A	(*no subclass*)	000
syntax error or access rule violation in dynamic SQL statement	37	(*no subclass*)	000
transaction rollback	40	(*no subclass*)	000
		integrity constraint violation	002
		serialization failure	001
		statement completion unknown	003
triggered data change violation	27	(*no subclass*)	000
warning	01	(*no subclass*)	000
		cursor operation conflict	001
		disconnect error	002
		implicit zero-bit padding	008

Table 23—SQLSTATE class and subclass values (Cont.)

Condition	Class	Subcondition	Subclass
		insufficient item descriptor areas	005
		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
with check option violation	44	(*no subclass*)	000

22.2 SQLCODE

Table 24, "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 24—SQLCODE values

Value	Condition
0	successful completion
+100	no data
−*n*	exception

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 American Standard. See Annex D, "Deprecated features".

23 Conformance

23.1 Introduction

This American 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".

23.2 Claims of conformance

Claims of conformance to this American Standard shall state:

1) Which *level of conformance* is claimed:
   a) Full SQL (The complete database language specified in this American 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) Which of the following *binding styles* are supported:
   a) Module (<module>)
   b) Embedded syntax (<embedded SQL host program>)
   c) Direct invocation and processing of SQL language (<direct SQL statement>)
3) For the binding styles module or embedded syntax, 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.

23.3 Extensions and options

A conforming implementation may provide additional facilities or options not specified by this American Standard. 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 American 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.

23.4 Flagger requirements

Implementations that claim conformance only to Entry SQL may, but are not required to, provide an SQL Flagger (see Subclause 4.34, "SQL Flagger").

Implementations that claim conformance to Intermediate SQL shall provide an SQL Flagger (see Subclause 4.34, "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 4.34, "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

23.5 Processing methods

This American Standard does not define the method by which an <embedded SQL host program> is processed. Although the processing of <embedded SQL host program> is defined in terms of derivation of a program compliant with a programming language standard and a <module>, implementations of SQL are not constrained to follow that method, provided that effect is achieved.

Although the processing of <direct SQL statement> is defined in terms of calls to <procedure>s in a <module>, implementations of Direct Invocation are not constrained to follow that method, so long as the same effect is achieved.

Annex A
(informative)

Leveling the SQL Language

This Annex describes the restrictions placed on conforming Intermediate SQL and Entry SQL language.

A.1 Intermediate SQL Specifications

1) Subclause 5.2, "<token> and <separator>":

   a) In conforming Intermediate SQL language, a <regular identifier> or a <delimited identifier body> shall not comprise more than 18 <character representation>s.

   b) A <delimiter token> shall not be a <bit string literal> or a <hex string literal>.

   c) Conforming Intermediate SQL language shall contain no <identifier body> that ends in an <underscore>.

2) Subclause 5.3, "<literal>":

   a) An <unsigned integer> that is a <seconds fraction> shall not contain more than 6 <digit>s.

   b) A <literal> or <unsigned literal> shall not be a <bit string literal> or a <hex string literal>.

   c) A <character string literal> shall not specify a <character set specification>.

3) Subclause 5.4, "Names and identifiers":

   a) Conforming Intermediate SQL language shall not contain any <extended statement name> or <extended cursor name>.

   b) Conforming Intermediate SQL language shall not contain any <collation name>.

   c) Conforming Intermediate SQL language shall not contain any explicit <catalog name>.

   d) Conforming Intermediate SQL language shall not contain any <translation name> or <form-of-use conversion name>.

   e) An <identifier> shall not specify a <character set specification>.

4) 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>.

5) Subclause 6.2, "<value specification> and <target specification>":

   a) In Intermediate SQL, the specific data type of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

6) Subclause 6.3, "<table reference>":

   a) A <table reference> shall not be a <derived table>.

7) Subclause 6.5, "<set function specification>":

   a) If a <general set function> specifies DISTINCT, then the <value expression> shall be a <column reference>.

8) Subclause 6.6, "<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 6.7, "<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 6.8, "<datetime value function>":

   a) Conforming Intermediate SQL language shall contain no <time precision> or <timestamp precision>.

11) Subclause 6.13, "<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> shall not contain more than one <row value constructor element>.

13) Subclause 7.2, "<table value constructor>":

   a) A <table value constructor> shall contain exactly one <row value constructor> that shall be of the form "(<value expression list>)".

b) A <table value constructor> shall be the <query expression> of an <insert statement>.

14) Subclause 7.3, "<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>.

15) Subclause 7.4, "<from clause>":

a) A <table reference> shall not be a <derived table>.

b) If the table identified by <table name> is a grouped view, then the <from clause> shall contain exactly one <table reference>.

16) Subclause 7.5, "<joined table>":

a) Conforming Intermediate SQL language shall contain no <cross join>.

b) Conforming Intermediate SQL language shall not specify UNION JOIN.

17) Subclause 7.7, "<group by clause>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

18) Subclause 7.9, "<query specification>":

a) The <key word> DISTINCT shall not be specified more than once in a <query specification>, excluding any <subquery> of that <query specification>.

b) If the <table expression> of the <query specification> is a grouped view, then the <select list> shall not contain a <set function specification>.

19) Subclause 7.10, "<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>.

20) Subclause 7.11, "<scalar subquery>, <row subquery>, and <table subquery>":

a) The <query expression> contained in a <subquery> shall be a <query specification>.

21) Subclause 8.1, "<predicate>":

a) A <predicate> shall not be a <match predicate>.

22) 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>.

23) Subclause 8.10, "<match predicate>":

a) Conforming Intermediate SQL language shall not contain any <match predicate>.

24) Subclause 8.12, "<search condition>":

a) A <boolean test> shall not specify a <truth value>.

25) Subclause 10.3, "<privileges>":

a) An <action> that specifies INSERT shall not contain a <privilege column list>.

26) Subclause 10.5, "<collate clause>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

27) Subclause 11.1, "<schema definition>":

a) A <schema element> shall not be an <assertion definition>.

b) Conforming Intermediate SQL language shall not contain any <collation definition>.

c) Conforming Intermediate SQL language shall not contain any <translation definition>.

28) Subclause 11.3, "<table definition>":

a) Conforming Intermediate SQL language shall not specify GLOBAL TEMPORARY and shall not reference any global temporary table.

29) Subclause 11.4, "<column definition>":

a) A <constraint attributes> shall not be specified, but INITIALLY IMMEDIATE and NOT DEFERRABLE are implicit.

b) Conforming Intermediate SQL language shall not contain any <collate clause>.

30) Subclause 11.6, "<table constraint definition>":

a) A <constraint attributes> shall not be specified, but INITIALLY IMMEDIATE and NOT DEFERRABLE are implicit.

31) Subclause 11.8, "<referential constraint definition>":

a) A <references specification> shall not specify MATCH.

b) A <referential triggered action> shall not contain an <update rule>.

32) Subclause 11.9, "<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.

33) Subclause 11.19, "<view definition>":

a) Conforming Intermediate SQL shall not contain any <levels clause>, but the effect shall be that defined for a <levels clause> of CASCADED.

34) Subclause 11.21, "<domain definition>":

a) Conforming Intermediate SQL language shall not contain any <collate clause>.

35) Subclause 11.22, "<alter domain statement>":

a) Conforming Intermediate SQL language shall contain no <alter domain statement>.

36) Subclause 11.23, "<set domain default clause>":

a) Conforming Intermediate SQL language shall contain no <set domain default clause>.

37) Subclause 11.24, "<drop domain default clause>":

a) Conforming Intermediate SQL language shall contain no <drop domain default clause>.

38) Subclause 11.25, "<add domain constraint definition>":

a) Conforming Intermediate SQL language shall contain no <add domain constraint definition>.

39) Subclause 11.26, "<drop domain constraint definition>":

a) Conforming Intermediate SQL language shall contain no <drop domain constraint definition>.

40) Subclause 11.30, "<collation definition>":

a) Conforming Intermediate SQL language shall not contain any <collation definition>.

41) Subclause 11.31, "<drop collation statement>":

a) Conforming Intermediate SQL language shall not contain any <drop collation statement>.

42) Subclause 11.32, "<translation definition>":

a) Conforming Intermediate SQL language shall contain no <translation definition>.

43) Subclause 11.33, "<drop translation statement>":

a) Conforming Intermediate SQL language shall contain no <drop translation statement>.

44) Subclause 11.34, "<assertion definition>":

a) Conforming Intermediate SQL language shall not contain any <assertion definition>.

45) Subclause 11.35, "<drop assertion statement>":

a) Conforming Intermediate SQL language shall not contain any <drop assertion statement>.

46) Subclause 11.36, "<grant statement>":

a) In Conforming Intermediate SQL language, an <object name> shall not specify COLLATION or TRANSLATION.

47) Subclause 12.1, "<module>":

a) A <module> shall not contain a <temporary table declaration>.

48) Subclause 12.3, "<procedure>":

a) A <parameter declaration> shall not specify a <data type> that is BIT or BIT VARYING.

49) Subclause 12.5, "<SQL procedure statement>":

a) An <SQL transaction statement> shall not be a <set constraints mode statement>.

b) An <SQL schema statement> shall not be an <assertion definition>.

c) An <SQL schema statement> shall not be a <drop assertion statement>.

d) An <SQL dynamic statement> shall not be an <allocate cursor statement> statement.

e) An <SQL session statement> shall not be a <set schema statement>, a <set catalog statement>, or a <set names statement>.

f) An <SQL schema statement> shall not be a <collation definition>, a <drop collation statement>, a <translation definition>, or a <drop translation statement>.

g) An <SQL dynamic statement shall not be a <deallocate prepared statement>, or a <describe input statement>.

h) An <SQL procedure statement> shall not be an <SQL connection statement>.

50) Subclause 13.1, "<declare cursor>":

a) Conforming Intermediate SQL language 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.

51) Subclause 13.3, "<fetch statement>":

a) A <fetch statement> shall not contain a <fetch orientation>.

52) Subclause 13.5, "<select statement: single row>":

a) The <table expression> shall not include a <group by clause> or a <having clause> and shall not identify a grouped view.

53) Subclause 13.6, "<delete statement: positioned>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <cursor specification> referenced by the <cursor name>.

54) Subclause 13.7, "<delete statement: searched>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <search condition>.

55) Subclause 13.8, "<insert statement>":

a) The base table that would be changed as a result of executing the <insert statement> shall not be identified in any <from clause> generally contained in the <query expression>.

56) Subclause 13.9, "<update statement: positioned>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <cursor specification> referenced by the <cursor name>.

57) Subclause 13.10, "<update statement: searched>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <search condition> or contained in the <value expression> of the <set clause>.

58) Subclause 13.11, "<temporary table declaration>":

a) Conforming Intermediate SQL language shall not contain any <temporary table declaration>.

59) Subclause 14.2, "<set constraints mode statement>":

a) Conforming Intermediate SQL language shall not contain any <set constraints mode statement>.

60) Subclause 15.1, "<connect statement>":

a) Conforming Intermediate SQL language shall not contain any <connect statement>.

61) Subclause 15.2, "<set connection statement>":

a) Conforming Intermediate SQL language shall not contain any <set connection statement>.

62) Subclause 15.3, "<disconnect statement>":

a) Conforming Intermediate SQL language shall not contain any <disconnect statement>.

63) Subclause 16.1, "<set catalog statement>":

a) Conforming Intermediate SQL language shall not contain any <set catalog statement>.

64) Subclause 16.2, "<set schema statement>":

a) Conforming Intermediate SQL language shall not contain any <set schema statement>.

65) Subclause 16.3, "<set names statement>":

a) Conforming Intermediate SQL language shall not contain any <set names statement>.

66) Subclause 17.2, "<allocate descriptor statement>":

a) An <occurrences> and a <descriptor name> shall be a <literal>.

67) Subclause 17.3, "<deallocate descriptor statement>":

a) A <descriptor name> shall be a <literal>.

68) Subclause 17.4, "<get descriptor statement>":

a) A <descriptor name> shall be a <literal>.

69) Subclause 17.5, "<set descriptor statement>":

a) A <descriptor name> shall be a <literal>.

70) Subclause 17.6, "<prepare statement>":

a) A <preparable SQL schema statement> shall not be an <assertion definition> or a <drop assertion statement>.

b) A <preparable SQL schema statement> shall not be a <collation definition>, a <drop collation statement>, a <translation definition>, or a <drop translation statement>.

c) A <preparable SQL session statement> shall not be a <set schema statement>.

d) A <preparable SQL session statement> shall not be a <set catalog statement>.

e) A <preparable SQL session statement> shall not be a <set names statement>.

f) A <preparable SQL transaction statement> shall not be a <set constraints mode statement>.

g) A <preparable SQL data statement> shall not be a <dynamic single row select statement>.

71) Subclause 17.7, "<deallocate prepared statement>":

a) Conforming Intermediate SQL language shall contain no <deallocate prepared statement>.

72) Subclause 17.8, "<describe statement>":

a) Conforming Intermediate SQL language shall not contain any <describe input statement>.

73) Subclause 17.9, "<using clause>":

a) A <descriptor name> shall be a <literal>.

74) Subclause 17.10, "<execute statement>":

a) Conforming Intermediate SQL language shall contain no <result using clause>.

75) Subclause 17.12, "<dynamic declare cursor>":

a) Conforming Intermediate SQL language shall contain no <dynamic declare cursor> that specifies 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.

76) Subclause 17.13, "<allocate cursor statement>":

a) Conforming Intermediate SQL language shall not contain any <allocate cursor statement>.

77) Subclause 17.15, "<dynamic fetch statement>":

a) A <dynamic fetch statement> shall not contain a <fetch orientation>.

78) Subclause 17.17, "<dynamic delete statement: positioned>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <cursor specification> referenced by the <dynamic cursor name>.

79) Subclause 17.18, "<dynamic update statement: positioned>":

a) Let *T* be the table identified by the <table name>. *T* shall not be a table that is identified in the <from clause> of any <subquery> contained in the <cursor specification> referenced by the <dynamic cursor name>.

80) Subclause 17.19, "<preparable dynamic delete statement: positioned>":

a) Conforming Intermediate SQL language shall contain no <preparable dynamic delete statement: positioned>.

81) Subclause 17.20, "<preparable dynamic update statement: positioned>":

a) Conforming Intermediate SQL language shall contain no <preparable dynamic update statement: positioned>.

82) Subclause 19.1, "<embedded SQL host program>":

a) An <embedded SQL statement> shall not contain a <temporary table declaration>.

83) Subclause 19.3, "<embedded SQL Ada program>":

a) An <Ada variable definition> shall not specify a bit string variable.

84) Subclause 19.4, "<embedded SQL C program>":

a) A <C derived variable> shall not be a <C bit variable>.

85) Subclause 19.5, "<embedded SQL COBOL program>":

a) A <COBOL type specification> shall not be a <COBOL bit type>.

86) Subclause 19.6, "<embedded SQL Fortran program>":

a) A <Fortran type specification> shall not specify BIT.

87) Subclause 19.8, "<embedded SQL Pascal program>":

a) A <Pascal type specification> shall not specify BIT or PACKED ARRAY [1..<length>] OF BIT.

88) Subclause 19.9, "<embedded SQL PL/I program>":

a) A <PL/I type specification> shall not specify BIT or BIT VARYING.

89) Subclause 20.1, "<direct SQL statement>":

a) A <direct SQL data statement> shall not be a <temporary table declaration>.

b) An <SQL session statement> shall not be a <set schema statement>, a <set catalog statement>, or a <set names statement>.

c) An <SQL transaction statement> shall not be a <set constraints mode statement>.

d) An <SQL schema statement> shall not be an <assertion definition> or a <drop assertion statement>.

e) An <SQL schema statement> shall not be a <collation definition>, a <drop collation statement>, a <translation definition> or a <drop translation statement>.

f) A <direct SQL statement> shall not be an <SQL connection statement>.

90) Subclause 21.2, "Information Schema":

a) Conforming Intermediate SQL language shall reference only the following Information Schema tables: SCHEMATA, DOMAINS, TABLES, VIEWS, COLUMNS, TABLE_PRIVILEGES, COLUMN_PRIVILEGES, USAGE_PRIVILEGES, REFERENTIAL_CONSTRAINTS, CHECK_CONSTRAINTS, KEY_COLUMN_USAGE, ASSERTIONS, CHARACTER_SETS, COLLATIONS, VIEW_COLUMN_USAGE, CONSTRAINT_TABLE_USAGE, CONSTRAINT_COLUMN_USAGE, SQL_LANGUAGES, COLUMN_DOMAIN_USAGE, and VIEW_TABLE_USAGE.

A.2 Entry SQL Specifications

1) All Intermediate SQL specifications are included as Entry SQL specifications.

2) Subclause 5.2, "<token> and <separator>":

a) A <delimiter token> shall not be a <national character string literal>.

b) An <identifier body> shall contain no <simple Latin lower case letter>.

3) Subclause 5.3, "<literal>":

a) A <literal> or <unsigned literal> shall not be a <national character string literal>.

b) A <literal> or <unsigned literal> shall not be a <datetime literal> or an <interval literal>.

4) Subclause 5.4, "Names and identifiers":

a) Conforming Entry SQL language shall not contain any <domain name>, <SQL statement name>, <dynamic cursor name>, or <constraint name>.

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 be a <dynamic parameter specification>.

b) A <general value specification> shall not specify VALUE.

c) A <general value specification> shall not specify SESSION_USER, SYSTEM_USER, or CURRENT_USER.

7) Subclause 6.3, "<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 6.5, "<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 6.6, "<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 6.7, "<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 6.8, "<datetime value function>":

   a) Conforming Entry SQL language shall not contain any <datetime value function>.

12) Subclause 6.9, "<case expression>":

   a) Conforming Entry SQL language shall not contain any <case expression>.

13) Subclause 6.10, "<cast specification>":

   a) Conforming Entry SQL language shall not contain any <cast specification>.

14) Subclause 6.11, "<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 6.13, "<string value expression>":

a) A <character value expression> shall not be a <concatenation>.

16) Subclause 6.14, "<datetime value expression>":

a) Conforming Entry SQL language shall not contain any <datetime value expression>.

17) Subclause 6.15, "<interval value expression>":

a) Conforming Entry SQL language shall not contain any <interval value expression>.

18) Subclause 7.1, "<row value constructor>":

a) A <value expression> that is contained in the <row value constructor> that is contained in the <table value constructor> that is contained in the <query expression> of an <insert statement> shall be a <value specification>.

b) A <row value constructor element> shall not specify DEFAULT.

19) Subclause 7.5, "<joined table>":

a) Conforming Entry SQL language shall not contain any <joined table>.

20) Subclause 7.6, "<where clause>":

a) A <value expression> directly contained in the <search condition> shall not include a reference to a column derived from a function.

21) Subclause 7.9, "<query specification>":

a) A <query specification> is not updatable if the <where clause> of the <table expression> contains a <subquery>.

b) A <select sublist> shall not specify "<qualifier>.*".

22) Subclause 7.10, "<query expression>":

a) A <query expression> shall not specify EXCEPT.

b) A <query term> shall not specify INTERSECT.

c) A <query primary> 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 descriptions of the first and second operands shall be identical and the description of the result is identical to the description of the operands.

23) Subclause 7.11, "<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.

24) 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>.

25) Subclause 8.5, "<like predicate>":

a) The <match value> shall be a <column reference>.

b) A <pattern> shall be a <value specification>.

c) An <escape character> shall be a <value specification>.

26) Subclause 8.6, "<null predicate>":

a) A <row value constructor> shall be a <column reference>.

27) Subclause 8.9, "<unique predicate>":

a) Conforming Entry SQL language shall not contain any <unique predicate>.

28) Subclause 8.11, "<overlaps predicate>":

a) Conforming Entry SQL language shall not contain any <overlaps predicate>.

29) Subclause 10.1, "<interval qualifier>":

a) Conforming Entry SQL language shall not contain any <interval qualifier>.

30) Subclause 10.2, "<language clause>":

a) Conforming Entry SQL language shall not contain a <language clause> that specifies MUMPS.

31) Subclause 10.4, "<character set specification>":

a) Conforming Entry SQL language shall not contain a <character set specification>.

32) Subclause 11.1, "<schema definition>":

a) A <schema element> shall not be a <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 Intermediate SQL language shall not contain any <character set definition>.

33) Subclause 11.2, "<drop schema statement>":

a) Conforming Entry SQL language shall not contain a <drop schema statement>.

34) Subclause 11.18, "<drop table statement>":

a) Conforming Entry SQL language shall not contain any <drop table statement>.

35) Subclause 11.4, "<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>.

36) Subclause 11.5, "<default clause>":

a) A <default option> shall not specify CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, SYSTEM_USER, SESSION_USER, or CURRENT_USER.

37) Subclause 11.6, "<table constraint definition>":

a) Conforming Entry SQL language shall not contain any <constraint name definition>.

38) Subclause 11.7, "<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.

39) Subclause 11.8, "<referential constraint definition>":

a) A <referential constraint definition> shall not contain a <referential triggered action>.

40) Subclause 11.10, "<alter table statement>":

a) Conforming Entry SQL language shall not contain an <alter table statement>.

41) Subclause 11.11, "<add column definition>":

a) Conforming Entry SQL language shall not contain an <add column definition>.

42) Subclause 11.12, "<alter column definition>":

a) Conforming Entry SQL language shall not contain an <alter column definition>.

43) Subclause 11.13, "<set column default clause>":

a) Conforming Entry SQL language shall not contain a <set column default clause>.

44) Subclause 11.14, "<drop column default clause>":

a) Conforming Entry SQL language shall not contain a <drop column default clause>.

45) Subclause 11.15, "<drop column definition>":

a) Conforming Entry SQL language shall not contain a <drop column definition>.

46) Subclause 11.16, "<add table constraint definition>":

a) Conforming Entry SQL language shall not contain an <add table constraint definition>.

47) Subclause 11.17, "<drop table constraint definition>":

a) Conforming Entry SQL language shall not contain a <drop table constraint definition>.

48) Subclause 11.19, "<view definition>":

a) The <query expression> in a <view definition> shall be a <query specification>.

49) Subclause 11.20, "<drop view statement>":

a) Conforming Entry SQL language shall not contain a <drop view statement>.

50) Subclause 11.36, "<grant statement>":

a) An <object name> shall not specify TABLE.

b) In Conforming Entry SQL language, an <object name> shall not specify CHARACTER SET or DOMAIN.

51) Subclause 11.37, "<revoke statement>":

a) Conforming Entry SQL language shall not contain a <revoke statement>.

52) Subclause 11.21, "<domain definition>":

a) Conforming Entry SQL language shall not contain any <domain definition>.

53) Subclause 11.27, "<drop domain statement>":

a) Conforming Entry SQL language shall not contain a <drop domain statement>.

54) Subclause 11.28, "<character set definition>":

a) Conforming Entry SQL language shall not specify any <character set definition>.

55) Subclause 11.29, "<drop character set statement>":

a) Conforming Entry SQL language shall contain no <drop character set statement>.

56) Subclause 12.1, "<module>":

a) A <module> shall be associated with an SQL-agent during its execution. An SQL-agent shall be associated with at most one <module>.

b) A <module contents> shall not be a <dynamic declare cursor>.

57) Subclause 12.2, "<module name clause>":

a) A <module character set specification> shall not be specified.

58) Subclause 12.3, "<procedure>":

a) A <parameter declaration> shall not specify a <data type> that is CHARACTER VARYING.

59) Subclause 12.5, "<SQL procedure statement>":

a) An <SQL procedure statement> shall not be an <SQL schema statement>.

b) An <SQL procedure statement> shall not be an <SQL dynamic statement>.

c) An <SQL session statement> shall not be a <set session authorization identifier statement> or a <set time zone statement>.

d) An <SQL transaction statement> shall not be a <set transaction statement>.

e) An <SQL procedure statement> shall not be an <SQL diagnostics statement>.

60) Subclause 13.1, "<declare cursor>":

a) A <declare cursor> shall not specify SCROLL.

b) A <cursor specification> shall not contain an <updatability clause>.

61) Subclause 13.3, "<fetch statement>":

a) A <fetch statement> shall not specify FROM.

b) 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.

62) Subclause 13.5, "<select statement: single row>":

a) 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.

63) Subclause 13.8, "<insert statement>":

a) The <query expression> that is contained in an <insert statement> shall be a <table value constructor> that specifies VALUES and contains exactly one <row value constructor> or it shall be a <query 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>.

64) 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.

65) 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*.

66) Subclause 14.1, "<set transaction statement>":

a) Conforming Entry SQL language shall not contain any <set transaction statement>.

67) Subclause 14.3, "<commit statement>":

a) In conforming Entry SQL language, WORK shall be specified.

68) Subclause 14.4, "<rollback statement>":

a) In conforming Entry SQL language, WORK shall be specified.

69) Subclause 16.4, "<set session authorization identifier statement>":

a) Conforming Intermediate SQL language shall not contain any <set session authorization identifier statement>.

70) Subclause 16.5, "<set local time zone statement>":

a) Conforming Entry SQL language shall not contain any <set local time zone statement>.

71) Subclause 17.2, "<allocate descriptor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

72) Subclause 17.3, "<deallocate descriptor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

73) Subclause 17.4, "<get descriptor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

74) Subclause 17.5, "<set descriptor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

75) Subclause 17.6, "<prepare statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

76) Subclause 17.7, "<deallocate prepared statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

77) Subclause 17.8, "<describe statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

78) Subclause 17.9, "<using clause>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

79) Subclause 17.10, "<execute statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

80) Subclause 17.11, "<execute immediate statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

81) Subclause 17.12, "<dynamic declare cursor>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

82) Subclause 17.13, "<allocate cursor statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

83) Subclause 17.14, "<dynamic open statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

84) Subclause 17.15, "<dynamic fetch statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

85) Subclause 17.16, "<dynamic close statement>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

86) Subclause 17.17, "<dynamic delete statement: positioned>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

87) Subclause 17.18, "<dynamic update statement: positioned>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

88) Subclause 17.19, "<preparable dynamic delete statement: positioned>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

89) Subclause 17.20, "<preparable dynamic update statement: positioned>":

a) Conforming Entry SQL language shall not contain any Dynamic SQL language.

90) Subclause 18.1, "<get diagnostics statement>":

a) Conforming Entry SQL language shall not contain any <get diagnostics statement>.

91) Subclause 19.1, "<embedded SQL host program>":

a) An <embedded SQL statement> shall not contain a <dynamic declare cursor>.

b) An <embedded SQL statement> shall not contain an <embedded character set declaration>.

92) Subclause 19.3, "<embedded SQL Ada program>":

a) An <Ada qualified type specification> shall not specify a <character set specification>.

93) Subclause 19.4, "<embedded SQL C program>":

a) A <C derived variable> shall not be a <C VARCHAR variable> or a <C extended char variable>.

94) Subclause 19.5, "<embedded SQL COBOL program>":

a) A <COBOL character type> shall not specify a <character set specification>.

b) A <COBOL integer type> shall not be a <COBOL binary integer>.

95) Subclause 19.6, "<embedded SQL Fortran program>":

a) A <Fortran type specification> shall not specify a <character set specification>.

96) Subclause 19.8, "<embedded SQL Pascal program>":

a) A <Pascal type specification> shall not specify a <character set specification>.

97) Subclause 19.9, "<embedded SQL PL/I program>":

a) A <PL/I type specification> shall not specify CHARACTER VARYING.

b) A <PL/I type specification> shall not specify a <character set specification>.

98) Subclause 20.1, "<direct SQL statement>":

a) A <direct SQL statement> shall not be an <SQL schema statement>.

b) An <SQL session statement> shall not be a <set session authorization identifier statement> or a <set local time zone statement>.

c) A <direct SQL data statement> shall not be a <temporary table declaration>.

99) Subclause 21.2, "Information Schema":

a) Conforming Entry SQL language shall not reference any views from the Information Schema.

Annex B
(informative)

Implementation-defined elements

This Annex references those features that are identified in the body of this American 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 4.4.1, "Characteristics of numbers": Whether truncation or rounding is performed when trailing digits are removed from a numeric value is implementation-defined.

3) Subclause 4.4.1, "Characteristics of 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 4.4.1, "Characteristics of numbers": It is implementation-defined which numerical values have approximations obtained by rounding or truncation for a given approximate numeric type.

5) Subclause 4.4.1, "Characteristics of numbers": The boundaries within which the normal rules of arithmetic apply are implementation-defined.

6) Subclause 4.6, "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.

7) Subclause 4.11, "SQL-schemas": The default schema for <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s is initially implementation-defined but may be changed by the use of <set schema statement>s.

8) Subclause 4.12, "Catalogs": The creation and destruction of catalogs is accomplished by implementation-defined means.

9) Subclause 4.12, "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.

10) Subclause 4.12, "Catalogs": The default catalog for <module>s whose <module authorization clause> does not specify an explicit <catalog name> to qualify <schema name> and the default catalog name substitution value for execution of <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s are implementation-defined.

11) Subclause 4.13, "Clusters of catalogs": A cluster is an implementation-defined collection of catalogs.

12) Subclause 4.13, "Clusters of catalogs": Whether or not any catalog can occur simultaneously in more than one cluster is implementation-defined.

13) Subclause 4.15, "SQL-environment": The constituents of an SQL-environment beyond those specified in Subclause 4.15, "SQL-environment", are implementation-defined.

14) Subclause 4.15, "SQL-environment": The rules determining whether a <module> is within the environment are implementation-defined.

15) Subclause 4.16, "Modules": The mechanisms by which <module>s are created or destroyed are implementation-defined.

16) Subclause 4.16, "Modules": The manner in which an association between a <module> and an SQL-agent is defined is implementation-defined.

17) Subclause 4.16, "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 4.18.1, "Status parameters": The negative (exception) values for the SQLCODE status parameter are implementation-defined.

19) Subclause 4.22.1, "Classes of SQL-statements": This American Standard permits implementations to provide additional, implementation-defined, statements that may fall into any of these categories; the classification of such statements is also implementation-defined.

20) Subclause 4.23, "Embedded syntax": Whether a portion of the name space is reserved by an implementation for the names of procedures, subroutines, program variables, branch labels, <module>s, or <procedure>s is implementation-defined; if a portion of the name space is so reserved, the portion reserved is also implementation-defined.

21) Subclause 4.25, "Direct invocation of SQL": The method of invoking <direct SQL statement>s, the method of raising conditions as a result of <direct SQL statement>s, the method of accessing diagnostic information, and the method of returning the results are all implementation-defined.

22) Subclause 4.28, "SQL-transactions": It is implementation-defined whether or not the non-dynamic or dynamic execution of an SQL-data statement or the execution of an <SQL dynamic data statement> is permitted to occur within the same SQL-transaction as the non-dynamic or dynamic execution of an SQL-schema statement. If it does occur, then the effect on any open cursor, prepared dynamic statement, or deferred constraint is also implementation-defined.

23) Subclause 4.28, "SQL-transactions": If an implementation detects unrecoverable errors and implicitly initiates the execution of a <rollback statement>, an exception condition is raised: *transaction rollback* with an implementation-defined subclass code.

24) Subclause 4.29, "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-environment and create an SQL-session.

25) Subclause 4.30, "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-environment is initiated. The default SQL-environment is implementation-defined.

26) Subclause 4.30, "SQL-sessions": The mechanism and rules by which an environment determines whether a call to a <procedure> is the last call within the last active <module> is implementation-defined.

27) Subclause 4.30, "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.

28) Subclause 4.30, "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 <module>s not having an explicit <module authorization identifier>.

29) Subclause 4.30, "SQL-sessions": When an SQL-session is initiated, there is an implementation-defined default catalog whose name is used to effectively qualify all unqualified <schema name>s contained in <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s or are contained in <direct SQL statement>s when those statements are invoked directly.

30) Subclause 4.30, "SQL-sessions": When an SQL-session is initiated, there is an implementation-defined default schema whose name is used to effectively qualify all unqualified <qualified name>s contained in <preparable statement>s that are dynamically prepared in the current SQL-session through the execution of <prepare statement>s and <execute immediate statement>s or are contained in <direct SQL statement>s when those statements are invoked directly.

31) Subclause 4.30, "SQL-sessions": When an SQL-session is initiated, there is an implementation-defined default character set that is used to identify the character set implicit for <identifier>s and <character string literal>s that are contained in <preparable statement>s when those statements are prepared in the current SQL-session by either an <execute immediate statement> or a <prepare statement> or are contained in <direct SQL statement>s when those statements are invoked directly.

32) Subclause 4.30, "SQL-sessions": When an SQL-session is initiated, there is an implementation-defined default time zone displacement that is used as the current default time zone displacement of the SQL-session.

33) Subclause 4.31, "Client-server operation": When an SQL-agent is active, it is bound in some implementation-defined manner to a single SQL-client.

34) Subclause 4.34, "SQL Flagger": If an SQL-implementation provides user options to process conforming SQL language in a nonconforming manner, 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.

35) Subclause 5.2, "<token> and <separator>": The end-of-line indicator (<newline>) is implementation-defined.

36) Subclause 5.3, "<literal>": The <character set name> character set used to represent national characters is implementation-defined.

37) 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.

38) 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.

39) Subclause 5.4, "Names and identifiers": The data type of the <simple value specification> of <extended statement name> shall be character string with an implementation-defined character set.

40) Subclause 5.4, "Names and identifiers": The data type of the <simple value specification> of <extended cursor name> shall be character string with an implementation-defined character set.

41) Subclause 5.4, "Names and identifiers": Those <identifier>s that are valid <authorization identifier>s are implementation-defined.

42) Subclause 5.4, "Names and identifiers": Those <identifier>s that are valid <catalog name>s are implementation-defined.

43) Subclause 5.4, "Names and identifiers": All <form-of-use conversion name>s are implementation-defined.

44) Subclause 6.1, "<data type>": The <character set name> associated with NATIONAL CHARACTER is implementation-defined.

45) Subclause 6.1, "<data type>": If a <precision> is omitted, then an implementation-defined <precision> is implicit.

46) Subclause 6.1, "<data type>": The decimal precision of a data type defined as DECIMAL for each value specified by <precision> is implementation-defined.

47) 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.

48) 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.

49) Subclause 6.1, "<data type>": The binary precision of a data type defined as FLOAT for each value specified by <precision> is implementation-defined.

50) Subclause 6.1, "<data type>": The precision of a data type defined as REAL is implementation-defined.

51) Subclause 6.1, "<data type>": The precision of a data type defined as DOUBLE PRECISION is implementation-defined, but greater than that for REAL.

52) Subclause 6.1, "<data type>": For every <data type>, the limits of the <data type> are implementation-defined.

53) 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.

54) Subclause 6.1, "<data type>": If CHARACTER SET is not specified for <character string type>, then the character set is implementation-defined.

55) 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.

56) 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.

57) Subclause 6.1, "<data type>": For the <approximate numeric type> FLOAT, the maximum value of <precision> is implementation-defined.

58) 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.

59) Subclause 6.1, "<data type>": The maximum value of <time fractional seconds precision> is implementation-defined, but shall not be less than 6.

60) Subclause 6.1, "<data type>": Interval arithmetic that involves leap seconds or discontinuities in calendars will produce implementation-defined results.

61) 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 length if it is variable-length or its length if it is fixed-length, are implementation-defined.

62) 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.

63) Subclause 6.2, "<value specification> and <target specification>": In Intermediate SQL, the specific data type of <indicator parameter>s and <indicator variable>s shall be the same implementation-defined data type.

64) Subclause 6.5, "<set function specification>": The precision of the value derived from application of the COUNT function is implementation-defined.

65) Subclause 6.5, "<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.

66) Subclause 6.5, "<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.

67) Subclause 6.5, "<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.

68) Subclause 6.6, "<numeric value function>": The precision of <position expression> is implementation-defined.

69) Subclause 6.6, "<numeric value function>": The precision of <extract expression> is implementation-defined. If <datetime field> specifies SECOND, then the scale is also implementation-defined.

70) Subclause 6.6, "<numeric value function>": The precision of <length expression> is implementation-defined.

71) Subclause 6.7, "<string value function>": The maximum length of <character translation> or <form-of-use conversion> is implementation-defined.

72) Subclause 6.10, "<cast specification>": Whether to round or truncate when casting to exact or approximate numeric data types is implementation-defined.

73) Subclause 6.12, "<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.

74) Subclause 6.12, "<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.

75) Subclause 6.12, "<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.

76) Subclause 6.12, "<numeric value expression>": Whether to round or truncate when performing division is implementation-defined.

77) Subclause 6.15, "<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.

78) 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.

79) 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.

80) Subclause 9.3, "Set operation result data types": If all of the data types in *DTS* are exact numeric, then the result data type is exact numeric with implementation-defined precision.

81) Subclause 9.3, "Set operation result data types": 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.

82) Subclause 10.1, "<interval qualifier>": The maximum value of <interval leading field precision> is implementation-defined, but shall not be less than 2.

83) Subclause 10.1, "<interval qualifier>": The maximum value of <interval fractional seconds precision> is implementation-defined, but shall not be less than 6.

84) Subclause 10.4, "<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.

85) Subclause 10.4, "<character set specification>": The default collating sequence of the character repertoire specified by an <implementation-defined character repertoire name> or an <implementation-defined universal character for-of-use name> and whether that collating sequence has the NO PAD attribute or the PAD SPACE attribute is implementation-defined.

86) 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.

87) Subclause 11.1, "<schema definition>": The privileges necessary to execute the <schema definition> are implementation-defined.

88) Subclause 11.30, "<collation definition>": The <standard collation name>s and <implementation-defined collation name>s that are supported are implementation-defined.

89) Subclause 11.30, "<collation definition>": The collating sequence resulting from the specification of EXTERNAL in a <collation definition> is implementation-defined.

90) Subclause 11.32, "<translation definition>": The <standard translation name>s and <implementation-defined translation name>s that are supported are implementation-defined.

91) Subclause 12.1, "<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 name clause>": 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.

93) Subclause 12.3, "<procedure>": Whether or not *SAI* can invoke <procedure>s in a <module> with explicit <module authorization identifier> *MAI* is implementation-defined, as are any restrictions pertaining to such invocation.

94) Subclause 12.3, "<procedure>": 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 <procedure> falls outside the set of values supported by the SQL-agent for that parameter, then the effect is implementation-defined.

95) Subclause 12.3, "<procedure>": If the <module> that contains the <procedure> is associated with an SQL-agent that is associated with another <module> that contains a <procedure> with the same <procedure name>, then the effect is implementation-defined.

96) Subclause 12.4, "Calls to a <procedure>": There is an implementation-defined package and character type for use in Ada bindings. The precisions and scales of the Ada types for SQL data types and the <exact numeric type> of indicator parameters are implementation-defined.

97) Subclause 12.4, "Calls to a <procedure>": The null character that defines the end of a C character string is implementation-defined.

98) Subclause 12.4, "Calls to a <procedure>": The number of bits in a C character is implementation-defined.

99) Subclause 12.4, "Calls to a <procedure>": The precision of an SQLCODE parameter in an <embedded COBOL program> is an implementation-defined value between 4 and 18, inclusive.

100) Subclause 12.4, "Calls to a <procedure>": The number of bits contained in a COBOL character is implementation-defined.

101) Subclause 12.4, "Calls to a <procedure>": The precision of a COBOL data type corresponding to SQL INTEGER or SMALLINT is implementation-defined.

102) Subclause 12.4, "Calls to a <procedure>": The number of bits contained in a Fortran character is implementation-defined.

103) Subclause 12.4, "Calls to a <procedure>": The number of bits contained in a Pascal character is implementation-defined.

104) Subclause 12.3, "<procedure>": The precision of an SQLCODE parameter in an <embedded PL/I program> is implementation-defined.

105) Subclause 12.4, "Calls to a <procedure>": The precision of a PL/I data type corresponding to SQL INTEGER or SMALLINT is implementation-defined.

106) 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.

107) Subclause 14.1, "<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 9, "SQL-transaction isolation levels and the three phenomena".

108) Subclause 14.3, "<commit statement>": If any error other than those specified in the General Rules 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.

109) Subclause 15.1, "<connect statement>": It is implementation-defined whether or not an SQL-implementation may support transactions that affect more than one SQL-server. If it does so, then the effects are implementation-defined.

110) Subclause 15.1, "<connect statement>": If <user name> is not specified, then an implementation-defined <user name> for the SQL-connection is implicit.

111) Subclause 15.1, "<connect statement>": The restrictions on whether or not the <user name> shall be identical to the <module authorization identifier> for the <module> that contains the <procedure> that contains the <connect statement> are implementation-defined.

112) Subclause 15.1, "<connect statement>": If DEFAULT is specified, then the method by which the default SQL-environment is determined is implementation-defined.

113) Subclause 15.1, "<connect statement>": The method by which <SQL-server name> is used to determine the appropriate SQL-environment is implementation-defined.

114) Subclause 16.4, "<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.

115) Subclause 17.2, "<allocate descriptor statement>": If WITH MAX <occurrences> is not specified, then an implementation-defined default value for <occurrences> that is greater than 0 is implicit.

116) Subclause 17.2, "<allocate descriptor statement>": The maximum number of SQL descriptor areas and the maximum number of item descriptor areas for a single SQL descriptor area are implementation-defined.

117) Subclause 17.5, "<set descriptor statement>": Restrictions on changing TYPE, LENGTH, PRECISION, DATETIME_INTERVAL_CODE, DATETIME_INTERVAL_PRECISION, SCALE, CHARACTER_SET_CATALOG, CHARACTER_SET_SCHEMA, and CHARACTER_SET_NAME values resulting from the execution of a <describe statement> before execution of a <execute statement>, <dynamic open statement>, or <dynamic fetch statement> are implementation-defined, except as specified in the General Rules of Subclause 17.9, "<using clause>".

118) Subclause 17.6, "<prepare statement>": The Format and Syntax Rules for a <preparable implementation-defined statement> are implementation-defined.

119) Subclause 18.1, "<get diagnostics statement>": The actual length of variable-length character items in the diagnostics area is implementation-defined but not less than 128.

120) Subclause 18.1, "<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'.

121) Subclause 18.1, "<get diagnostics statement>": The value of MESSAGE_TEXT is an implementation-defined character string.

122) Subclause 19.1, "<embedded SQL host program>": If an <embedded character set declaration> is not specified, then an <embedded character set declaration> containing an implementation-defined <character set specification> is implicit.

123) Subclause 19.3, "<embedded SQL Ada program>": If <character set specification> is not specified when CHAR is specified, then an implementation-defined <character set specification> is implicit.

124) Subclause 19.3, "<embedded SQL Ada program>": SQLCODE_TYPE describes an exact numeric variable whose precision is the implementation-defined precision defined for the SQLCODE parameter.

125) Subclause 19.4, "<embedded SQL C program>": If <character set specification> is not specified when <C character variable> or <C VARCHAR variable> is specified, then an implementation-defined <character set specification> is implicit.

126) Subclause 19.5, "<embedded SQL COBOL program>": The COBOL data description clauses, in addition to the PICTURE, SIGN, USAGE and VALUE clauses, that may appear in a <COBOL variable definition> are implementation-defined.

127) Subclause 19.5, "<embedded SQL COBOL program>": If <character set specification> is not specified when <COBOL character type> is specified, then an implementation-defined <character set specification> is implicit.

128) Subclause 19.5, "<embedded SQL COBOL program>": The precision of the <COBOL type specification> that corresponds to SQLCODE is implementation-defined.

129) Subclause 19.6, "<embedded SQL Fortran program>": If <character set specification> is not specified when CHARACTER is specified, then an implementation-defined <character set specification> is implicit.

130) Subclause 19.7, "<embedded SQL MUMPS program>": In a <MUMPS character variable>, an implementation-defined <character set specification> is implicit.

131) Subclause 19.8, "<embedded SQL Pascal program>": If <character set specification> is not specified when PACKED ARRAY OF CHAR is specified, then an implementation-defined <character set specification> is implicit.

132) Subclause 19.9, "<embedded SQL PL/I program>": The PL/I data description clauses, in addition to the <PL/I type specification> and the INITIAL clause, that may appear in a <PL/I variable definition> are implementation-defined.

133) Subclause 19.9, "<embedded SQL PL/I program>": If <character set specification> is not specified when CHARACTER is specified, then an implementation-defined <character set specification> is implicit.

134) Subclause 19.9, "<embedded SQL PL/I program>": The precision of the <PL/I type specification> that corresponds to SQLCODE is implementation-defined.

135) Subclause 20.1, "<direct SQL statement>": The <value specification> that represents the null value is implementation-defined.

136) Subclause 20.1, "<direct SQL statement>": The Format, Syntax Rules, and Access Rules for <direct implementation-defined statement> are implementation-defined.

137) Subclause 20.1, "<direct SQL statement>": Whether a <direct implementation-defined statement> may be associated with an active transaction is implementation-defined.

138) Subclause 20.1, "<direct SQL statement>": Whether a <direct implementation-defined statement> initiates a transaction is implementation-defined.

139) Subclause 20.1, "<direct SQL statement>": The methods of raising a condition and of accessing diagnostics information are implementation-defined.

140) Subclause 20.2, "<direct select statement: multiple rows>": 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.

141) Subclause 20.2, "<direct select statement: multiple rows>": The method of returning the results is implementation-defined.

142) Subclause 21.3.13, "TABLE_CONSTRAINTS base table": If the containing <add table constraint definition> does not specify a <constraint name definition>, then the values of CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, and CONSTRAINT_NAME are implementation-defined.

143) Subclause 21.3.23, "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.

144) Subclause 21.3.24, "COLLATIONS base table": The value of PAD_ATTRIBUTE for the collation SQL_TEXT is implementation-defined.

145) Subclause 21.3.26, "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.

146) Subclause 22.1, "SQLSTATE": The character set associated with the class value and subclass value of the SQLSTATE parameter is implementation-defined.

147) Subclause 22.1, "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.

148) Subclause 22.2, "SQLCODE": The negative values returned in an SQLCODE parameter to indicate exception conditions are implementation-defined.

149) Clause 23, "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 American 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 3.3.3, "Specification of the Information Schema" The actual objects on which the Information Schema views are based are implementation-dependent.

2) Subclause 3.3.4.1, "Exceptions": 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 American Standard, is implementation-dependent.

3) Subclause 3.3.4.1, "Exceptions": 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.

4) Subclause 3.3.4.3, "Terms denoting rule requirements": The treatment of language that does not conform to the SQL Formats and Syntax Rules is implementation-dependent.

5) Subclause 3.3.4.4, "Rule evaluation order": 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.

6) Subclause 3.3.4.4, "Rule evaluation order": 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.

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 for each data type is implementation-dependent.

9) Subclause 4.9, "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.

10) Subclause 4.9, "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.

11) Subclause 4.9, "Tables": Whether a temporary viewed table is materialized is implementation-dependent.

12) Subclause 4.19, "Diagnostics area": The actual size of the diagnostics area is implementation-dependent when the SQL-agent does not specify the size.

13) Subclause 4.19, "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.

14) Subclause 4.21, "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.

15) Subclause 4.21, "Cursors": When the ordering of a cursor is not defined by an <order by clause>, the relative position of two rows is implementation-dependent.

16) Subclause 4.21, "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.

17) Subclause 4.21, "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, then whether that change will be visible through that cursor before it is closed is implementation-dependent.

18) Subclause 4.26, "Privileges": The mapping of <authorization identifier>s to operating system users is implementation-dependent.

19) Subclause 4.26, "Privileges": 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>.

20) Subclause 4.27, "SQL-agents": An SQL-agent is an implementation-dependent entity that causes the execution of SQL-statements.

21) Subclause 4.28, "SQL-transactions": The schema definitions that are implicitly read on behalf of executing an SQL-statement are implementation-dependent.

22) Subclause 4.30, "SQL-sessions": The SQL-session <module> contains a <module authorization clause> that specifies SCHEMA <schema name>, where the value of <schema name> is implementation-dependent.

23) Subclause 4.30, "SQL-sessions": A unique implementation-dependent SQL-session identifier is associated with each SQL-session.

24) Subclause 4.31, "Client-server operation": The <module name> of the <module> that is effectively materialized on an SQL-server is implementation-dependent.

25) Subclause 4.31, "Client-server operation": Diagnostic information is passed to the diagnostics area in the SQL-client is passed in an implementation-dependent manner.

26) Subclause 4.31, "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 6.4, "<column reference>": The implicit qualifier of a <column reference> for which there is more than one possible qualifier with most local scope is implementation-dependent.

28) Subclause 6.8, "<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.

29) Subclause 6.15, "<interval value expression>": The start datetime used for converting intervals to scalars for subtraction purposes is implementation-dependent.

30) 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.

31) Subclause 7.9, "<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.

32) Subclause 7.10, "<query expression>": If a <simple table> is not a <query specification>, then the name of each column of the <simple table> is implementation-dependent.

33) Subclause 7.10, "<query expression>": 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>.

34) Subclause 7.10, "<query expression>": 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>.

35) Subclause 8.2, "<comparison predicate>": 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*.

36) Subclause 8.2, "<comparison predicate>": When the operations MAX, MIN, DISTINCT, references to 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.

37) Subclause 9.3, "Set operation result data types": 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*.

38) Subclause 11.6, "<table constraint definition>": The <constraint name> of a constraint that does not specify a <constraint name definition> is implementation-dependent.

39) Subclause 11.8, "<referential constraint definition>": The specific value to use for cascading among various values that are not distinct is implementation-dependent.

40) Subclause 11.21, "<domain definition>": The <constraint name> of a constraint that does not specify a <constraint name definition> is implementation-dependent.

41) Subclause 12.1, "<module>": If the SQL-agent that performs a call of a <procedure> in a <module> is not a standard program in the language specified in the <language clause> of the <module>, then the results are implementation-dependent.

42) Subclause 12.1, "<module>": If the SQL-agent that performs a call of a <procedure> in a <module> is not a program that conforms to the programming language standard specified by the <language clause> of that <module>, then the effect is implementation-dependent.

43) Subclause 12.3, "<procedure>": The <procedure name> should be a standard-conforming procedure, function, or routine name of the language specified by the subject <language clause>. Failure to observe this recommendation will have implementation-dependent effects.

44) Subclause 12.3, "<procedure>": After the execution of the last <procedure>, if an unrecoverable error has not occurred, and the SQL-agent did not terminate unexpectedly, and no constraint is not satisfied, then the choice of whether to perform a <commit statement> or a <rollback statement> is implementation-dependent. The determination of whether an SQL-agent has terminated unexpectedly is implementation-dependent.

45) Subclause 12.3, "<procedure>": If there are more than one status parameter, then the order in which values are assigned to these status parameters is implementation-dependent.

46) Subclause 13.1, "<declare cursor>": If a <declare cursor> does not contain an <order by clause>, then the ordering of rows in the table associated with that <declare cursor> is implementation-dependent.

47) 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 are the null value or 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.

48) 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.

49) Subclause 13.3, "<fetch statement>": If an error occurs during assignment of a value to a target during the execution of a <fetch statement>, then the values of targets other than status parameters are 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>": 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.

52) 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.

53) Subclause 14.1, "<set transaction statement>": If <number of conditions> is not specified, then an implementation-dependent value not less than 1 is implicit.

54) 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.

55) Subclause 17.4, "<get descriptor statement>": If an exception condition is raised in a <get descriptor statement>, then the values of all targets specified by <simple target specification 1> and <simple target specification 2> are implementation-dependent.

56) Subclause 17.4, "<get descriptor statement>": For a <select list> column, if the column name is implementation-dependent, then NAME is the implementation-dependent name for the column and UNNAMED is set to 1.

57) Subclause 17.4, "<get descriptor statement>": For a <dynamic parameter specification>, the values of NAME and UNNAMED are implementation-dependent.

58) Subclause 17.5, "<set descriptor statement>": Item descriptor area fields not relevant to the data type of the item being described according to the General Rules of Subclause 17.5, "<set descriptor statement>", are set to implementation-dependent values.

59) Subclause 17.5, "<set descriptor statement>": If an exception condition is raised in a <set descriptor statement>, then the values of all elements of the descriptor specified in the <set descriptor statement> are implementation-dependent.

60) Subclause 17.6, "<prepare statement>": The validity of an <extended statement name> value or a <statement name> in an SQL-transaction different from the one in which the statement was prepared is implementation-dependent.

61) Subclause 17.9, "<using clause>": When a <describe output statement> is executed, the values of DATA and INDICATOR, as well as the value of other fields not relevant to the data type of the described item, are implementation-dependent. If the column name is implementation-dependent, then NAME is set to that implementation-dependent name.

62) Subclause 17.9, "<using clause>": When a <describe input statement> is used, the values for NAME, DATA, and INDICATOR, as well as the value of other fields not relevant to the data type of the described item, in the SQL dynamic descriptor area structure are implementation-dependent.

63) Subclause 18.1, "<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.

64) Subclause 18.1, "<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.

65) Subclause 19.1, "<embedded SQL host program>": The <module name> of the implied <module> derived from an <embedded SQL host program> is implementation-dependent.

66) Subclause 19.1, "<embedded SQL host program>": The <module authorization identifier> of the implied <module> derived from an <embedded SQL host program> is implementation-dependent.

67) Subclause 19.1, "<embedded SQL host program>": In each <declare cursor> in the implied <module> derived from an <embedded SQL host program>, each <embedded variable name> has been replaced consistently with a distinct <parameter name> that is implementation-dependent.

68) Subclause 19.1, "<embedded SQL host program>": The <procedure name> of each <procedure> in the implied <module> derived from an <embedded SQL host program> is implementation-dependent.

69) Subclause 19.1, "<embedded SQL host program>": In each <procedure> in the implied <module> derived from an <embedded SQL host program>, each <embedded variable name> has been replaced consistently with a distinct <parameter name> that is implementation-dependent.

70) Subclause 19.1, "<embedded SQL host program>": For <SQL procedure statement>s other than <open statement>s, whether one <procedure> in the implied <module> derived from an <embedded SQL host program> can correspond to more than one <SQL procedure statement> in the <embedded SQL host program> is implementation-dependent.

71) Subclause 19.1, "<embedded SQL host program>": In each <procedure> in the implied <module> derived from an <embedded SQL host program>, the order of the instances of <parameter declaration> is implementation-dependent.

72) Subclause 20.1, "<direct SQL statement>": If an unrecoverable error has occurred, or if the direct invocation of SQL 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. The determination of whether a direct invocation of SQL has terminated unexpectedly is implementation-dependent.

73) Subclause 20.2, "<direct select statement: multiple rows>": If an <order by clause> is not specified, then the ordering of rows in the table associated with that <declare cursor> is implementation-dependent.

74) Subclause 20.2, "<direct select statement: multiple rows>": If an <order by clause> is specified and a group of two or more rows in the resulting table contain values that are the null value or 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.

75) Subclause 21.3.23, "CHARACTER_SETS base table": The values of DEFAULT_COLLATE_SCHEMA, DEFAULT_COLLATE_CATALOG, and DEFAULT_COLLATE_NAME for default collations specifying the order of characters in a repertoire are 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 American Standard:

1) SQLCODE

2) <COBOL computational integer>s in <embedded SQL COBOL program>s.

3) The <unsigned integer> option of <sort specification> of <declare cursor>.

4) The <parameter declaration>s in a <procedure> without enclosing parentheses and without commas separating multiple <parameter declaration>s.

Annex E
(informative)

Incompatibilities with ANSI X3.135-1989

This American Standard introduces some incompatibilities with the earlier version of Database Language SQL as specified in ANSI X3.135-1989. Unless specified in this Annex, features and capabilities of Database Language SQL are compatible with the earlier version of this American Standard.

1) In Subclause 4.12, "Cursors", and Subclause 8.3, "<declare cursor>", General Rule 3)a)iii) of ANSI X3.135-1989, a requirement was made that multiple openings of an unordered cursor within a single SQL-transaction return the rows in the same order every time. In this American Standard, the order of rows returned for an unordered cursor is always implementation-dependent.

2) In ANSI X3.135-1989, <parameter name>s were simple <identifier>s. In this American Standard, colons are now required to delimit <parameter name>s.

3) In ANSI X3.135-1989, the specification of the enforcement of WITH CHECK OPTION for a view was ambiguous. This American Standard clarifies the intent of WITH CHECK OPTION.

4) In ANSI X3.135-1989, the <unique constraint definition> had no rule to prevent defining two unique constraints with identical unique column lists on a table. This American Standard adds a Syntax Rule to require that all unique column lists be distinct.

5) In ANSI X3.135-1989, there were no requirements that users have SELECT privileges on tables that were referenced in contexts other than <select statement: single row> or <fetch statement>. This American Standard adds Access Rules to require that users have SELECT privilege on tables referenced in various <query expression>s, <table expression>s, <search condition>s, and <value expression>s.

6) In ANSI X3.135-1989, the specification of a view did not preclude the possibility of defining a view in terms of itself. This American Standard adds a Syntax Rule in Subclause 11.19, "<view definition>" to preclude that possibility.

7) In ANSI X3.135-1989, the specification did not preclude the possibility of using outer references in <set function specification>s in <subquery>s contained in the <search condition> of a <having clause>, but did not define the semantics of such a construction. This American Standard adds a Syntax Rule in Subclause 6.5, "<set function specification>", and a Syntax Rule in Subclause 7.6, "<where clause>", to preclude that possibility.

8) In ANSI X3.135-1989, the <module> derived from an <embedded SQL host program> had an implementor-defined <module authorization identifier>. As the time of binding was not specified, this meant that the authorization identifier for privilege checking could be determined at either compile-time or run-time. In this American Standard, the derived <module> has no <module authorization identifier> and the authorization identifier for privilege checking is determined at run-time.

9) In ANSI X3.135-1989, if a cursor is on or before a row and that row is deleted, then the cursor is positioned before the row that is immediately after the position of the deleted row. In this American Standard, if the deletion is not made through that cursor, then the effect of the deletion is implementation-dependent.

10) A number of additional <reserved word>s have been added to the language. These <reserved word>s are:

— ABSOLUTE

— ACTION

— ADD

— ALLOCATE

— ALTER

— ARE

— ASSERTION

— AT

— BETWEEN

— BIT

— BIT_LENGTH

— BOTH

— CASCADE

— CASCADED

— CASE

— CAST

— CATALOG

— CHAR_LENGTH

— CHARACTER_LENGTH

— COALESCE

— COLLATE

— COLLATION

— COLUMN

— CONNECT

— CONNECTION

— CONSTRAINT

— CONSTRAINTS

— CONVERT

— CORRESPONDING

— CROSS

— CURRENT_DATE

— CURRENT_TIME

— CURRENT_TIMESTAMP

— CURRENT_USER

— DATE

— DAY

— DEALLOCATE

— DEFERRABLE

— DEFERRED

— DESCRIBE

— DESCRIPTOR

— DIAGNOSTICS

— DISCONNECT

— DOMAIN

— DROP

— ELSE

— END-EXEC

— EXCEPT

— EXCEPTION

— EXECUTE

— EXTERNAL

— EXTRACT

— FALSE
— FIRST
— FULL
— GET
— GLOBAL
— HOUR
— IDENTITY
— IMMEDIATE
— INITIALLY
— INNER
— INPUT
— INSENSITIVE
— INTERSECT
— INTERVAL
— ISOLATION
— JOIN
— LAST
— LEADING
— LEFT
— LEVEL
— LOCAL
— LOWER
— MATCH
— MINUTE
— MONTH
— NAMES
— NATIONAL
— NATURAL
— NCHAR
— NEXT

— NO

— NULLIF

— OCTET_LENGTH

— ONLY

— OUTER

— OUTPUT

— OVERLAPS

— PAD

— PARTIAL

— POSITION

— PREPARE

— PRESERVE

— PRIOR

— READ

— RELATIVE

— RESTRICT

— REVOKE

— RIGHT

— ROWS

— SCROLL

— SECOND

— SESSION

— SESSION_USER

— SIZE

— SPACE

— SQLSTATE

— SUBSTRING

— SYSTEM_USER

— TEMPORARY

— THEN

— TIME
— TIMESTAMP
— TIMEZONE_HOUR
— TIMEZONE_MINUTE
— TRAILING
— TRANSACTION
— TRANSLATE
— TRANSLATION
— TRIM
— TRUE
— UNKNOWN
— UPPER
— USAGE
— USING
— VALUE
— VARCHAR
— VARYING
— WHEN
— WRITE
— YEAR
— ZONE

Annex F
(informative)

Maintenance and interpretation of SQL

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 X3.135-1989, the following items have resulted in formal interpretations of Database Language SQL.

1) Expression Evaluation

2) Read-Only Cursors

3) Working Viewed Tables

4) Authorization Id

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 #1 (SQLIB-1) published April 19, 1991, by Global Engineering Documents, Inc.

Since publication of ANSI X3.135-1989, 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 9075:1989. Numbers in parentheses refer to JTC1/SC21/WG3 documents that identify the defects.

1) General Clarifications, Defect Report 9075/5 (WG3 N1150, N1160)

2) General Clarifications, Defect Report 9075/6 (WG3 N1161)

The SQL language corrections proposed in each defect report were accepted by SC21/WG3 in May, 1991 (see SC21 N6249, Arles 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 9075:1992 as a replacement standard for ISO/IEC 9075:1989 and by adoption of ANSI X3.135-1992 as a replacement standard for ANSI X3.135-1989.

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)
311 First St. NW, Suite 500,
Washington, DC 20001.

Index

— B —

— C —

— E —

— F —

— G —

— H —

— I —

— J —

— K —

— L —

— M —

— N —

— O —

— P —

— Q —

— R —

— T —

— U —

— V —

— W —

— Y —

— Z —