ANSI X3.135-1986

American National Standard

Adopted for Use by
the Federal Government

FIPS PUB 127

See Notice on Inside
Front Cover

for information systems –

database language –
SQL

american national standards institute, inc.
1430 broadway, new york, new york 10018

American National Standard

Approval of an American National Standard requires verification by ANSI that the requirements for due process, consensus, and other criteria for approval have been met by the standards developer.

Consensus is established when, in the judgment of the ANSI Board of Standards Review, substantial agreement has been reached by directly and materially affected interests. Substantial agreement means much more than a simple majority, but not necessarily unanimity. Consensus requires that all views and objections be considered, and that a concerted effort be made toward their resolution.

The use of American National Standards is completely voluntary; their existence does not in any respect preclude anyone, whether he has approved the standards or not, from manufacturing, marketing, purchasing, or using products, processes, or procedures not conforming to the standards.

The American National Standards Institute does not develop standards and will in no circumstances give an interpretation of any American National Standard. Moreover, no person shall have the right or authority to issue an interpretation of an American National Standard in the name of the American National Standards Institute. Requests for interpretations should be addressed to the secretariat or sponsor whose name appears on the title page of this standard.

CAUTION NOTICE: This American National Standard may be revised or withdrawn at any time. The procedures of the American National Standards Institute require that action be taken to reaffirm, revise, or withdraw this standard no later than five years from the date of approval. Purchasers of American National Standards may receive current information on all standards by calling or writing the American National Standards Institute.

This standard has been adopted for Federal Government use.

Details concerning its use within the Federal Government are contained in Federal Information Processing Standards Publication 127, Database Language SQL. For a complete list of the publications available in the Federal Information Processing Standards Series, write to the Standards Processing Coordinator (ADP), Institute for Computer Sciences and Technology, National Bureau of Standards, Gaithersburg, MD 20899.

Published by

American National Standards Institute
1430 Broadway, New York, New York 10018

Printed in the United States of America

PC2M687/22

ANSI®
X3.135-1986

American National Standard for Information Systems –

Database Language – SQL

Secretariat

Computer and Business Equipment Manufacturers Association

Approved October 16, 1986

American National Standards Institute, Inc

Foreword

(This Foreword is not part of American National Standard X3.135-1986.)

American National Standard Database Language SQL specifies the syntax and semantics of interfaces to a database management system for defining and accessing SQL databases. Together, these interfaces are called Database Language SQL.

This standard was developed by the Technical Committee on Database, X3H2, under project 363D authorized by the Accredited National Standards Committee on Information Processing Systems, X3. SPARC document number 81-689 describes the purpose of this project as follows:

> "Develop a standard for the functionality of the interfaces (i.e. the set of functions and the semantics of individual functions) to [a relational database management system]. These functions will be used in defining, querying, and altering relational databases."

American National Standard Database Language SQL is the culmination of work by many groups. During work on the SQL standard, Technical Committee X3H2 addressed all proposals that were submitted, whether for addition, deletion, or change.

This standard was approved as an American National Standard by the American National Standards Institute on October 16, 1986.

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

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

E. Lohse, Chair
R. Gibson, Vice-Chair
C. A. Kachurik, Administrative Secretary

Organization Represented	**Name of Representative**
American Express	D. L. Seigal
	L. Durfee (Alt)
American Library Association	P. Peters
American Nuclear Society	G. C. Main
	D. R. Vondy (Alt)
AMP Inc.	P. E. Lannan
	E. Kelly (Alt)
Association for Computing Machinery	K. Magel
	J. A. Meads (Alt)
Association of the Institute for Certification of Computer Professionals	T. M. Kurihara
AT&T Communications	H. L. Marchese

Organization Represented	Name of Representative
AT&T Technologies	R. Gibson (Alt)
	H. V. Bertine
	P. D. Bartoli (Alt)
	S. M. Garland (Alt)
Burroughs Corporation	S. Fenner
Control Data Corporation	C. E. Cooper
	K. A. Lucke (Alt)
Cooperating Users of Burroughs Equipment	T. Easterday
	D. Miller (Alt)
Data General Corporation	J. Pilat
	L. Chapin (Alt)
Data Processing Management Association	C. G. Meyer
	W. Arrington (Alt)
	T. H. Felker (Alt)
Digital Equipment Computer Users Society	W. Hancock
	D. Perry (Alt)
Digital Equipment Corporation	G. S. Robinson
	D. L. Shoemaker (Alt)
Eastman Kodak	G. Haines
	C. C. Bard (Alt)
General Electric Company	R. W. Signor
	W. R. Kruesi (Alt)
General Services Administration	W. C. Rinehuls
	L. L. Jackson (Alt)
GUIDE International	F. Kirshenbaum
	S. S. Abraham (Alt)
Harris Corporation	W. G. Fredrickson
	R. Sinha (Alt)
Hewlett-Packard	D. C. Loughry
Honeywell Information Systems	T. J. McNamara
	D. M. Taylor (Alt)
IBM Corporation	M. A. Gray
	R. H. Follett (Alt)
IEEE Computer Society	S. I. Sherr
	T. M. Kurihara (Alt)
	T. A. Varetoni (Alt)
Lawrence Berkeley Laboratory	D. F. Stevens
	R. L. Fink (Alt)
Moore Business Forms	D. H. Oddy
National Bureau of Standards	R. E. Rountree
	J. H. Burrows (Alt)
National Communications System	G. W. White
NCR Corporation	T. W. Kern
	A. R. Daniels (Alt)
Prime Computer, Inc.	J. Schmidt
	J. McHugh (Alt)
Railinc Corporation	R. A. Petrash
Recognition Technology Users Association	H. F. Schantz
	G. W. Wetzel (Alt)
Scientific Computer Systems Corporation	J. A. Baker
	C. Haberland (Alt)
SHARE, Inc.	T. B. Steel

Organization Represented	Name of Representative
	R. P. Rannie (Alt)
Sperry Corporation	M. W. Bass
	J. G. Smith (Alt)
Texas Instruments, Inc.	P. Smith
	R. F. Trow, Jr (Alt)
3M Company	P. D. Jahnke
	J. W. Van Valkenburg (Alt)
Travelers Insurance Companies, Inc.	J. T. Brophy
U. S. Department of Defense	F. Virtue
	B. Leong-Hong (Alt)
VIM	C. Tanner
	M. Sparks (Alt)
VISA USA	J. T. McKenna
	S. Crawford (Alt)
Wang Laboratories, Inc.	M. Hayek
	J. St. Amand (Alt)
Xerox Corporation	J. L. Wheeler
	R. Pierce (Alt)

Technical Committee X3H2 on Database, which developed this standard, had the following members:

Donald Deutsch, Chair
Oris Friesen, Vice-Chair
Michael Gorman, Secretary
Leonard Gallagher, International Representative

Dean Anderson	Wally Gazdik	Mark Lipp
Jerry Baker	Adrian Gonzalez	Phil Neches
Nick Baxter	Stephen Hollander	Ken Paris
Alan Bier	Andrew Hutt	Phil Shaw
Jarvis Boykin	Carol Joyce	Val Skalabrin
Jeannette Duffy	Sue Karlin	Jagan Sud
David Dul	Michael Kelley	Joan Sullivan
Andrew Eisenberg	Steve Klein	Ken Szczesny
Gerald Feldman	Bernard Kocis	Barry Vickers
Lynn Francis	May Kovalick	Elaine Volkman
Glen Fullmer	Dennis Leatherwood	Jerry Wisdorf

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

Ellen Boughter	Jack Jones	John Robertson
Fritz Bryant	Michael Kay	Tom Rogers
Gary Gregory	Stefan Langsner	Lawrence Scearce, Jr
Katherine Hammer	Frank Manola	Willem Stoeller
Rita Hillyer	Anthony Marriott	Michael Thibado
Gary James	Larry Pelletier	
Jane Jodiet	Marie Pierce	

Contents

American National Standard for Information Systems –

Database Language – SQL

1. Scope and field of application

This standard specifies the syntax and semantics of two database languages:

1) A schema definition language (SQL-DDL), for declaring the structures and integrity constraints of an SQL database.

2) A module language and a data manipulation language (SQL-DML), for declaring the database procedures and executable statements of a specific database application program.

This standard defines the logical data structures and basic operations for an SQL database. It provides functional capabilities for designing, accessing, maintaining, controlling, and protecting the database.

This standard provides a vehicle for portability of database definitions and application programs between conforming implementations.

This standard specifies two levels. Level 2 is the complete SQL database language. Level 1 is the subset of Level 2 defined in clause 9, "Levels" on page 87.

NOTE: Additional SQL language is planned for later addenda to this standard. Major topics under consideration for such addenda include referential integrity, enhanced transaction management, specification of certain implementor-defined rules, enhanced character handling facilities, and support for national character sets.

Annexes to this standard specify embedded syntax for including SQL data manipulation language statements in an otherwise standard application program. Such embedded syntax is defined to be a shorthand notation for a standard application program in which the embedded SQL statements have been replaced with explicit "calls" of database procedures that contain the SQL statements.

This standard applies to implementations that exist in an environment that may include 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. References

2.1 American National Standards

This standard is intended for use with the following American National Standards. When these standards are superceded by revisions approved by the American National Standards Institute, the revisions shall apply.

ANSI X3.9-1978, ISO 1539-1980, *Programming Language - FORTRAN.*

ANSI X3.23-1985, ISO 1989-1985, *Programming Language - COBOL.*

ANSI X3.53-1976, ISO 6160-1979, *Programming Language - PL/I.*

2.2 Other Standards

This standard is also intended for use with *Programming Language - Pascal* [1], BSI BS 6192-1982, ISO 7185-1983.

---

[1] This British Standard is available from the American National Standards Institute, 1430 Broadway, New York, N.Y. 10018.

3. Overview

3.1 Organization

The organization of this standard is as follows:

1) 3.2, "Notation" on page 5 and 3.3, "Conventions" on page 5 define the notations and conventions used in this standard.

2) 3.4, "Conformance" on page 6 defines conformance criteria.

3) Clause 4, "Concepts" on page 9 defines terms and presents concepts used in the definition of SQL.

4) Clause 5, "Common elements" on page 15 defines language elements that occur in several parts of SQL language.

5) Clause 6, "Schema definition language" on page 53 defines the SQL facilities for specifying a database.

6) Clause 7, "Module language" on page 61 defines SQL modules and procedures.

7) Clause 8, "Data manipulation language" on page 67 defines the data manipulation statements of SQL.

8) Clause 9, "Levels" on page 87 defines the two levels of SQL.

3.2 Notation

The syntactic notation used in this standard is BNF ("Backus Normal Form", or "Backus-Naur Form"), with the following extensions:

1) Square brackets ([]) indicate optional elements.

2) Ellipses (...) indicate elements that may be repeated one or more times.

3) Braces ({}) group sequences of elements.

In the BNF syntax, a production symbol <A> is defined to "contain" a production symbol <B> if <B> occurs someplace in the expansion of <A>. If <A> contains <B>, then <B> is "contained in" <A>. If <A> contains <B>, then <A> is the "containing" <A> production symbol for <B>.

3.3 Conventions

Syntactic elements of this standard are specified in terms of:

1) *Function*: A short statement of the purpose of the element.

2) *Format*: A BNF definition of the syntax of the element.

3) *Syntax Rules*: Additional syntactic constraints not expressed in BNF that the element shall satisfy.

4) *General Rules*: A sequential specification of the run-time effect of the element.

In the Syntax Rules, the term "shall" defines conditions that are required to be true of syntactically conforming SQL language. The treatment of SQL language that does not conform to the Formats or the Syntax Rules is implementor-defined.

In the General Rules, the term "shall" defines conditions that are tested at run-time during the execution of SQL statements. If all such conditions are true, then the statement executes successfully and the SQLCODE parameter is set to a defined nonnegative number. If any such condition is false, then the statement does not execute successfully, the statement execution has no effect on the database, and the SQLCODE parameter is set to an implementor-defined negative number.

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 the database as that sequence. The term "effectively" is used in the General Rules to emphasize actions whose effect might be achieved in other ways by an implementation.

The term "persistent object" is used to characterize objects such as <module>s and <schema>s that are created and destroyed by implementor-defined mechanisms.

In this standard, clauses begin a new odd-numbered page, and in clause 5, "Common elements" on page 15 through clause 8, "Data manipulation language" on page 67 subclauses begin a new page. The resulting blank space is not significant.

3.4 Conformance

This standard specifies conforming SQL language and conforming SQL implementations. Conforming SQL language shall abide by the BNF Format and associated Syntax Rules. A conforming SQL implementation shall process standard conforming SQL language according to the General Rules.

An implementation claiming SQL-DDL conformance shall process SQL-DDL (<schema>) at level 1 or level 2.

An implementation claiming SQL-DML conformance shall process, either at level 1 or level 2:

1) Direct invocation of SQL data manipulation language statements (<SQL statement>); and/or

2) Module language (<module>); and/or

3) one or more of

   a) Embedded SQL COBOL (<embedded SQL COBOL program>);

   b) Embedded SQL FORTRAN (<embedded SQL FORTRAN program>);

   c) Embedded SQL Pascal (<embedded SQL Pascal program>);

   d) Embedded SQL PL/I (<embedded SQL PL/I program>).

An implementation claiming full SQL conformance shall provide, either at level 1 or level 2, SQL-DDL conformance and SQL-DML conformance.

A conforming implementation may provide additional facilities or options not specified by this 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.

Claims of conformance to this standard shall state:

1) Which of the following types of conformance are claimed:

   a) Full SQL conformance to level 1;

   b) Full SQL conformance to level 2;

   c) SQL-DDL conformance to level 1;

   d) SQL-DDL conformance to level 2;

   e) SQL-DML conformance to level 1;

   f) SQL-DML conformance to level 2.

2) Which of the following facilities are implemented:

   a) Direct processing of SQL data manipulation language statements;

   b) Module language (<module>);

   c) Embedded SQL COBOL (<embedded SQL COBOL program>);

   d) Embedded SQL FORTRAN (<embedded SQL FORTRAN program>);

   e) Embedded SQL Pascal (<embedded SQL Pascal program>);

   f) Embedded SQL PL/I (<embedded SQL PL/I program>).

This standard does not define the method or the time of binding between application programs and database management system components.

4. Concepts

4.1 Sets

A set is an unordered collection of distinct objects.

A multi-set is an unordered collection of objects that are not necessarily distinct.

A sequence is an ordered collection of objects that are not necessarily distinct.

The cardinality of a collection is the number of objects in that collection. Unless specified otherwise, any collection may be empty.

4.2 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 implementor-defined.

A value is primitive, in that it has no logical subdivision within this standard. A value is a null value or a nonnull value.

A null value is an implementor-defined type-dependent special value that is distinct from all nonnull values of that type.

A nonnull value is either a character string or a number. A character string and a number are not comparable values.

4.2.1 Character strings

A character string consists of a sequence of characters of the implementor-defined character set. A character string has a length, which is a positive integer that specifies the number of characters in the sequence.

All character strings are comparable. A character string is identical to another character string if and only if it is equal to that character string in accordance with the comparison rules specified in 5.11, "<comparison predicate>" on page 32.

4.2.2 Numbers

A number is either an exact numeric value or an approximate numeric value. All numbers are comparable values.

An exact numeric value has a precision and a scale. The precision is a positive integer that determines the number of significant decimal digits. The scale is a nonnegative integer. A scale of 0 indicates that the number is an integer. For a scale of N, the exact numeric value is the integer value of the significant digits multiplied by 10 to the power -N.

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.

Whenever an exact 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 is represented in the data type of the target. The value is converted to have the precision and scale of the target.

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.

4.3 Columns

A column is a multi-set of values that may vary over time. All values of the same column are of the same data type 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.

A column has a description and an ordinal position within a table. The description of a column includes its data type and an indication of whether the column is constrained to contain only nonnull values. The description of a character string column specifies its length attribute. The description of an approximate numeric column specifies the precision of its numbers. The description of an exact numeric column specifies the precision and scale of its numbers.

A named column is a column of a named table or a column that inherits the description of a named column. The description of a named column includes its name.

4.4 Tables

A table is a multi-set 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 has a description. The description includes a description of each of its columns.

A base table is a named table defined by a <table definition>. The description of a base table includes its name.

A derived table is a table derived directly or indirectly from one or more other tables by the evaluation of a <query specification>. The values of a derived table are those of the underlying tables when it is derived.

A viewed table is a named derived table defined by a <view definition>. The description of a viewed table includes its name.

A table is either updatable or read-only. The operations of insert, update, and delete are permitted for updatable tables and are not permitted for read-only tables.

A grouped table is a set of groups derived during the evaluation of a <group by clause>. A group is a multi-set of rows in which all values of the grouping column(s) are equal. A grouped table may be considered as a collection of tables. Set functions may operate on the individual tables within the grouped table.

A grouped view is a viewed table derived from a grouped table.

4.5 Integrity constraints

Integrity constraints define the valid states of the database by constraining the values in the base tables. Constraints may be defined to prevent two rows in a table from having the same values in a specified column or columns (UNIQUE) or to prevent a column from containing a null value (NOT NULL).

Integrity constraints are effectively checked after execution of each <SQL statement>. If the base table associated with an integrity constraint does not satisfy that integrity constraint, then the <SQL statement> has no effect and the SQLCODE parameter is set to an implementor-defined negative number.

4.6 Schemas

A <schema> is a persistent object specified by the schema definition language. It consists of a <schema authorization clause> and all <table definition>s, <view definition>s, and <privilege definition>s known to the system for a specified <authorization identifier> in an environment. The concept of environment is implementor-defined.

The tables, views, and privileges defined by a <schema> are considered to be "owned by" or to have been "created by" the <authorization identifier> specified for that <schema>.

NOTE: An implementation may provide facilities (such as DROP TABLE, DROP VIEW, ALTER TABLE, and REVOKE) that allow the definitions of the tables, views, and privileges for a given <authorization identifier> to be created, destroyed, and modified incrementally over time. This standard, however, only addresses the <schema>s that represent the definitions known to the system at a given time.

4.7 The database

The database is the collection of all data defined by the <schema>s in an environment. The concept of environment is implementor-defined.

4.8 Modules

A <module> is a persistent object specified in the module language. A <module> consists of an optional <module name>, a <language clause>, a <module authorization clause>, zero or more cursors specified by <declare cursor>s, and one or more <procedure>s.

An application program is a segment of executable code, possibly consisting of multiple subprograms. A single <module> is associated with an application program during its execution. An application program shall be associated with at most one <module>. The manner in which this association is specified, including the possible requirement for execution of some implementor-defined statement, is implementor-defined.

4.9 Procedures

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

An application program associated with a <module> may reference the <procedure>s of that <module> by a "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 statement> that it contains to be executed.

4.10 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>.

4.10.1 SQLCODE parameter

The SQLCODE parameter is a special integer parameter. Its value is set to a status code that either indicates that a call of the <procedure> completed successfully or that an exception condition occurred during execution of the <procedure>.

4.10.2 Indicator parameters

An indicator parameter is an integer parameter that is specified after another parameter. Its primary use is to indicate whether the value that the other parameter assumes or supplies is a null value.

4.11 Standard programming languages

This standard specifies the actions of <procedure>s in <module>s when those <procedure>s are called by programs that conform to specified standard programming languages. The terms "standard COBOL program", "standard FORTRAN program", "standard Pascal program", and "standard PL/I program" refer to programs that meet the conformance criteria of the standards listed in clause 2, "References" on page 3.

4.12 Cursors

A cursor is specified by a <declare cursor>.

For each <declare cursor> in a <module>, a cursor is effectively created when a transaction (see 4.16, "Transactions" on page 14) referencing the <module> is initiated, and destroyed when that transaction is terminated.

A cursor is in either the open state or the closed state. The initial state of a cursor is the closed state. A cursor is placed in the open state by an <open statement> and returned to the closed state by a <close statement>, a <commit statement>, or a <rollback statement>.

A cursor in the open state designates a table, an ordering of the rows of that table, and a position relative to that ordering. If the <declare cursor> does not specify an <order by clause>, then the rows of the table have

an implementor-defined order. This order is subject to the reproducibility requirement within a transaction (see 4.16, "Transactions" on page 14), but it may change between transactions.

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 even though the table is empty.

A <fetch statement> advances the position of an open cursor to the next row of the cursor's ordering and retrieves the values of the columns of that row. An <update statement: positioned> updates the current row of the cursor. A <delete statement: positioned> deletes the current row of the cursor.

If a cursor is before a row and a new row is inserted at that position, then the effect, if any, on the position of the cursor is implementor-defined.

If a cursor is on a row 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. If such a row does not exist, then the position of the cursor is after the last row.

If an error occurs during the execution of an <SQL statement> that identifies an open cursor, then the effect, if any, on the position or state of that cursor is implementor-defined.

A working table is a table resulting from the opening of a cursor. Whether opening a cursor results in creation of a working base table or a working viewed table is implementor-defined.

Each row of a working viewed table is derived only when the cursor is positioned on that row.

A working base table is created when the cursor is opened and destroyed when the cursor is closed.

4.13 Statements

An <SQL statement> specifies a database operation or a cursor operation. A <select statement> fetches values from a table. An <insert statement> inserts rows into a table. A <update statement: searched> or <update statement: positioned> updates the values in rows of a table. A <delete statement: searched> or <delete statement: positioned> deletes rows of a table.

4.14 Embedded syntax

An <embedded SQL host program> (<embedded SQL COBOL program>, <embedded SQL FORTRAN program>, <embedded SQL Pascal program>, or <embedded SQL PL/I program>) is an application program 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. This allows database applications to be expressed in a hybrid form in which <SQL statement>s are embedded directly in an application program. Such a hybrid application program is defined to be equivalent to a standard application program in which the <SQL statement>s have been replaced by standard procedure or subroutine CALLs of SQL <procedure>s in a separate SQL <module>.

4.15 Privileges

A privilege authorizes a given category of <action> to be performed on a specified table or view by a specified <authorization identifier>. The <action>s that can be specified are INSERT, DELETE, SELECT, and UPDATE.

An <authorization identifier> is specified for each <schema> and <module>.

The <authorization identifier> specified for a <schema> shall be different from the <authorization identifier> of any other <schema> in the same environment. The <authorization identifier> of a <schema> is the "owner" of all tables and views defined in that <schema>.

Tables and views are designated by <table name>s. A <table name> consists of an <authorization identifier> and an <identifier>. The <authorization identifier> identifies the <schema> in which the table or view designated by the <table name> was defined. Tables and views defined in different <schema>s can have the same <identifier>.

If a reference to a <table name> does not explicitly contain an <authorization identifier>, then the <authorization identifier> of the containing <schema> or <module> is specified by default.

The <authorization identifier> of a <schema> has all privileges on the tables and views defined in that <schema>.

A <schema> with a given <authorization identifier> may contain <privilege definition>s that grant privileges to other <authorization identifier>s. The granted privileges may apply to tables and views defined in the current <schema>, or they may be privileges that were granted to the given <authorization identifier> by other <schema>s. The WITH GRANT OPTION clause of a <privilege definition> specifies whether the recipient of a privilege may grant it to others.

A <module> specifies an <authorization identifier>, the <module authorization identifier>, which shall have the privileges specified for each <SQL statement> in the <module>.

4.16 Transactions

A transaction is a sequence of operations, including database operations, that is atomic with respect to recovery and concurrency. A transaction is initiated when a <procedure> is called and no transaction is currently active. A transaction is terminated by a <commit statement> or a <rollback statement>. If a transaction is terminated by a <commit statement>, then all changes made to the database by that transaction are made accessible to all concurrent transactions. If a transaction is terminated by a <rollback statement>, then all changes made to the database by that transaction are canceled. Committed changes cannot be canceled. Changes made to the database by a transaction can be perceived by that transaction, but until that transaction terminates with a <commit statement> they cannot be perceived by other transactions.

The execution of concurrent transactions is guaranteed to be serializable. A serializable execution is defined to be an execution of the operations of concurrently executing transactions that produces the same effect as some serial execution of those same transactions. A serial execution is one in which each transaction executes to completion before the next transaction begins.

The execution of an <SQL statement> within a transaction has no effect on the database other than the effect stated in the General Rules for that <SQL statement>. Together with serializable execution, this implies that all read operations are reproducible within a transaction, except for changes explicitly made by the transaction itself.

5. Common elements

5.1 <character>

Function

Define the terminal symbols of the language and the elements of strings.

Format

<character> ::=
<digit> | <letter> | <special character>

<digit> ::=
0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<letter> ::=
<upper case letter> | <lower case letter>

<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

<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

<special character> ::=
See Syntax Rule 1.

Syntax Rules

1) A <special character> is any character in the implementor-defined character set other than a <digit> or a <letter>. If the implementor-defined end-of-line indicator is a character, then it is also excluded from <special character>.

   NOTE: See the Format for <newline> in 5.3, "<token>" on page 18.

2) The <special character>s shall include all characters other than <digit>s or <letter>s that occur in terminal productions of SQL language, and shall include the percent sign and underscore characters.

General Rules

None.

5.2 <literal>

Function

Specify a nonnull value.

Format

```
<literal> ::=
                <character string literal>
              | <numeric literal>

<character string literal> ::=
              '<character representation>...'

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

<nonquote character> ::=
              See Syntax Rule 1.

<quote representation> ::=
              ''

<numeric literal> ::=
                <exact numeric literal>
              | <approximate numeric literal>

<exact numeric literal> ::=
              [+|-] {   <unsigned integer>[.<unsigned integer>]
                      | <unsigned integer>.
                      | .<unsigned integer>}

<approximate numeric literal> ::=
              <mantissa>E<exponent>

<mantissa> ::= <exact numeric literal>

<exponent> ::= <signed integer>

<signed integer> ::= [+|-] <unsigned integer>

<unsigned integer> ::=
              <digit>...
```

Syntax Rules

1) A <nonquote character> is any <character> other than the single quote mark character (').

2) The data type of a <character string literal> is character string. The length of a <character string literal> is the number of <character representation>s that it contains. Each <quote representation> in a <character string literal> represents a single quotation mark character in both the value and the length of the <character string literal>.

3) An <exact numeric literal> without a decimal point (.) has an implied decimal point following the last <digit>.

4) 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 decimal point.

5) The data type of an <approximate numeric literal> is approximate numeric. The precision of an <approximate numeric literal> is the precision of its <mantissa>.

General Rules

1) The value of a <character string literal> is the sequence of <character>s that it contains.

2) The numeric value of an <exact numeric literal> is derived from the normal mathematical interpretation of signed positional decimal notation.

3) The numeric value of an <approximate numeric literal> is the product of the exact numeric value represented by the <mantissa> with the number obtained by raising the number 10 to the power represented by the <exponent>.

5.3 <token>

Function

Specify lexical units.

Format

```
<token> ::=
            <nondelimiter token> | <delimiter token>

<nondelimiter token> ::=
             <identifier>
            | <key word>
            | <numeric literal>

<identifier> ::=
            <upper case letter> [{[<underscore>] <letter or digit>}...]

<underscore> ::= _

<letter or digit> ::=
            <upper case letter> | <digit>

<key word> ::=
             ALL | AND | ANY | AS | ASC | AUTHORIZATION | AVG
            | BEGIN | BETWEEN | BY
            | CHAR | CHARACTER | CHECK | CLOSE | COBOL | COMMIT
            | CONTINUE | COUNT | CREATE | CURRENT | CURSOR
            | DEC | DECIMAL | DECLARE | DELETE | DESC | DISTINCT | DOUBLE
            | END | ESCAPE | EXEC | EXISTS
            | FETCH | FLOAT | FOR | FORTRAN | FOUND | FROM
            | GO | GOTO | GRANT | GROUP | HAVING
            | IN | INDICATOR | INSERT | INT | INTEGER | INTO | IS
            | LANGUAGE | LIKE
            | MAX | MIN | MODULE | NOT | NULL | NUMERIC
            | OF | ON | OPEN | OPTION | OR | ORDER
            | PASCAL | PLI | PRECISION | PRIVILEGES | PROCEDURE | PUBLIC
            | REAL | ROLLBACK
            | SCHEMA | SECTION | SELECT | SET | SMALLINT | SOME
            | SQL | SQLCODE | SQLERROR | SUM
            | TABLE | TO | UNION | UNIQUE | UPDATE | USER
            | VALUES | VIEW | WHENEVER | WHERE | WITH | WORK

<delimiter token> ::=
            <character string literal>
            |,|(|)|<|>|.|:|=|*|+|-|/|<>|>=|<=

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

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

<newline> ::=
implementor-defined end-of-line indicator

<space> ::=
space character

Syntax Rules

1) A <token>, other than a <character string literal>, shall not include a <space>.

2) Any <token> may be followed by a <separator>. A <nondelimiter token> shall be followed by a <delimiter token> or a <separator>. If the syntax does not allow a <nondelimiter token> to be followed by a <delimiter token>, then that <nondelimiter token> shall be followed by a <separator>.

3) An <identifier> shall not consist of more than 18 <character>s.

4) An <identifier> shall not be identical to a <key word>.

General Rules

None.

5.4 Names

Function

Specify names.

Format

<table name> ::= [<authorization identifier>.] <table identifier>

<authorization identifier> ::= <identifier>

<table identifier> ::= <identifier>

<column name> ::= <identifier>

<correlation name> ::= <identifier>

<module name> ::= <identifier>

<cursor name> ::= <identifier>

<procedure name> ::= <identifier>

<parameter name> ::= <identifier>

Syntax Rules

1) A <table name> identifies a named table.

2) If a <table name> does not contain an <authorization identifier>, then:

   Case:

   a) If the <table name> is contained in a <schema>, then the <authorization identifier> specified as the <schema authorization identifier> of the <schema> is implicit.

   b) If the <table name> is contained in a <module>, then the <authorization identifier> specified as the <module authorization identifier> of the <module> is implicit.

3) Two <table name>s are equal if and only if they have the same <table identifier> and the same <authorization identifier>, regardless of whether the <authorization identifier>s are implicit or explicit.

4) A <table name> is declared in a <table definition> or <view definition>.

5) A <table name> in an <SQL statement> shall identify a table defined in the <schema>.

6) An <authorization identifier> represents an authorization identifier.

7) An <identifier> is declared as a <correlation name> and associated with a table for a particular scope. The scope of a <correlation name> is either a <select statement>, <subquery>, or <query specifica-

tion> (see 5.20, "<from clause>" on page 43). Scopes may be nested. In different scopes, the same <correlation name> may be associated with different tables or with the same table.

8) A <column name> identifies a named column. An <identifier> is defined as a <column name> by a <table definition> or <view definition>.

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

10) A <cursor name> identifies a <cursor>.

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

12) A <parameter name> identifies a parameter.

General Rules

None.

5.5 <data type>

Function

Specify a data type.

Format

```
<data type> ::=
              <character string type>
            | <exact numeric type>
            | <approximate numeric type>

<character string type> ::=
              CHARACTER [(<length>)]
            |  CHAR [(<length>)]

<exact numeric type> ::=
              NUMERIC [(<precision> [,<scale>])]
            | DECIMAL [(<precision> [,<scale>])]
            | DEC [(<precision> [,<scale>])]
            | INTEGER
            | INT
            | SMALLINT

<approximate numeric type> ::=
              FLOAT [(<precision>)]
            | REAL
            | DOUBLE PRECISION

<length> ::= <unsigned integer>

<precision> ::= <unsigned integer>

<scale> ::= <unsigned integer>
```

Syntax Rules

1) CHAR is a synonym for CHARACTER. DEC is a synonym for DECIMAL. INT is a synonym for INTEGER.

2) The value of an <unsigned integer> that is a <length> or a <precision> shall be greater than 0.

3) If a <length> is omitted, then it is assumed to be 1. If a <scale> is omitted, then it is assumed to be 0. If a <precision> is omitted, then it is implementor-defined.

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

5) CHARACTER specifies the data type character string, with length specified by the <length>.

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

7) DECIMAL specifies the data type exact numeric, with scale specified by the <scale> and with implementor-defined precision equal to or greater than the value of the specified <precision>.

8) INTEGER specifies the data type exact numeric, with implementor-defined precision and scale 0.

9) SMALLINT specifies the data type exact numeric, with scale 0 and implementor-defined precision that is not larger than the implementor-defined precision of INTEGER.

10) FLOAT specifies the data type approximate numeric, with binary precision equal to or greater than the value of the specified <precision>.

11) REAL specifies the data type approximate numeric, with implementor-defined precision.

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

General Rules

None.

5.6 <value specification> and <target specification>

Function

Specify one or more values, parameters, or variables.

Format

```
<value specification> ::=
        <parameter specification>
      | <variable specification>
      | <literal>
      | USER

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

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

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

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

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

Syntax Rules

1) A <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. The data type of an indicator parameter shall be exact numeric with a scale of 0. The specific <exact numeric type> of indicator parameters is implementor-defined.

3) A <variable specification> identifies a host variable or a host variable and an indicator variable. The data type of an indicator variable shall be the implementor-defined data type specified for indicator parameters.

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

5) A <parameter specification> shall be contained in a <module>. A <variable specification> shall be contained in an <embedded SQL statement>.

6) The data type of USER is character string of implementor-defined length.

General Rules

1) 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>.

2) 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 <variable name>.

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

4) The value specified by USER is equal to the <authorization identifier> specified as the <module authorization identifier> of the <module> that contains the <SQL statement> whose execution caused the USER <value specification> to be evaluated.

5.7 <column specification>

Function

Reference a named column.

Format

<column specification> ::=
[<qualifier>.] <column name>

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

Syntax Rules

1) A <column specification> references a named column. The meaning of a reference to a column depends on the context.

2) Let C be the <column name> of the <column specification>.

3) Case:

   a) If a <column specification> contains a <qualifier>, then the <column specification> shall appear within the scope of one or more <table name>s or <correlation name>s equal to that <qualifier>. If there is more than one such <table name> or <correlation name>, then the one with the most local scope is specified. The table associated with the specified <table name> or <correlation name> shall include a column whose <column name> is C.

   b) If a <column specification> does not include a <qualifier>, then it shall be contained within the scope of one or more <table name>s or <correlation name>s. Of these, let the phrase "possible qualifiers" denote those <table name>s and <correlation name>s whose associated table includes a column whose <column name> is C. There shall be exactly one possible qualifier with the most local scope, and that <table name> or <qualifier> is implicitly specified.

   NOTE: The "scope" of a <table name> or <correlation name> is specified in 5.20, "<from clause>" on page 43, 8.5, "<delete statement: searched>" on page 73, 8.11, "<update statement: positioned>" on page 83, and 8.12, "<update statement: searched>" on page 85.

4) If a <column specification> is contained in a <table expression> T and the scope of the implicitly or explicitly specified <qualifier> of the <column specification> is some <SQL statement> or <table expression> that contains the <table expression> T, then the <column specification> is an "outer reference" to the table associated with that <qualifier>.

5) Let T denote the table associated with the explicitly or implicitly specified <qualifier> R. The data type of a <column specification> is the data type of column C of T.

General Rules

1) "C" or "R.C" references column C in a given row of T.

5.8 <set function specification>

Function

Specify a value derived by the application of a function to an argument.

Format

<set function specification> ::=
COUNT(*) | <distinct set function> | <all set function>

<distinct set function> ::=
{AVG | MAX | MIN | SUM | COUNT} (DISTINCT <column specification>)

<all set function> ::=
{AVG | MAX | MIN | SUM} ([ALL] <value expression>)

Syntax Rules

1) The argument of COUNT(*) and the argument source of a <distinct set function> and <all set function> is a table or a group of a grouped table as specified in 5.19, "<table expression>" on page 42, 5.24, "<subquery>" on page 48, and 5.25, "<query specification>" on page 50.

2) Let R denote the argument or argument source of a <set function specification>.

3) The <column specification> of a <distinct set function> and each <column specification> in the <value expression> of an <all set function> shall unambiguously reference a column of R and shall not reference a column derived from a <set function specification>.

4) The <value expression> of an <all set function> shall include a <column specification> that references a column of R and shall not include a <set function specification>. If the <column specification> is an outer reference, then the <value expression> shall not include any operators.

   NOTE: "Outer reference" is defined in 5.7, "<column specification>" on page 26.

5) If a <set function specification> contains a <column specification> that is an outer reference, then the <set function specification> shall be contained in a <subquery> of a <having clause>.

   NOTE: "Outer reference" is defined in 5.7, "<column specification>" on page 26.

6) Let T be the data type of the values that result from evaluation of the <column specification> or <value expression>.

7) If COUNT is specified, then the data type of the result of a <set function specification> is exact numeric with implementor-defined precision and scale 0.

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

9) If SUM or AVG is specified, then:

   a) T shall not be character string.

b) If SUM is specified and T is exact numeric with scale S, then the data type of the result is exact numeric with implementor-defined precision and scale S.

c) If AVG is specified and T is exact numeric, then the data type of the result is exact numeric with implementor-defined precision and scale.

d) If T is approximate numeric, then the data type of the result is approximate numeric with implementor-defined precision.

General Rules

1) The argument of a <distinct set function> is a set of values. The set is derived by the elimination of any null values and any redundant duplicate values from the column of R referenced by the <column specification>.

2) The argument of an <all set function> is a multi-set of values. The multi-set is derived by the elimination of any null values from the result of the application of the <value expression> to each row of R. The specification or omission of ALL does not affect the meaning of an <all set function>.

3) Let S denote the argument of a <distinct set function> or an <all set function>.

4) Case:

a) If the <distinct set function> COUNT is specified, then the result is the cardinality of S.

b) If COUNT(*) is specified, then the result is the cardinality of R.

c) If AVG, MAX, MIN, or SUM is specified and S is empty, then the result is the null value.

d) If MAX or MIN is specified, then the result is respectively the maximum or minimum value in S. These results are determined using the comparison rules specified in 5.11, "<comparison predicate>" on page 32.

e) If SUM is specified, then the result is the sum of the values in S. The sum shall be within the range of the data type of the result.

f) If AVG is specified, then the result is the average of the values in S. The sum of the values in S shall be within the range of the data type of the result.

5.9 <value expression>

Function

Specify a value.

Format

```
<value expression> ::=
                <term>
              | <value expression> + <term>
              | <value expression> - <term>

<term> ::=
                <factor>
              | <term> * <factor>
              | <term> / <factor>

<factor> ::=
             [+ | -] <primary>

<primary> ::=
                <value specification>
              | <column specification>
              | <set function specification>
              | (<value expression>)
```

Syntax Rules

1) A <value expression> that includes a <distinct set function> shall not include any dyadic operators.

2) The first <character> of the <token> following a monadic operator shall not be a plus or minus sign.

3) If the data type of a <primary> is character string, then the <value expression> shall not include any operators. The data type of the result is character string.

4) If the data type of both operands of an 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 implementor-defined, and the scale is max(s1, s2).

   c) The precision of the result of multiplication is implementor-defined, and the scale is s1+s2.

   d) The precision and scale of the result of division is implementor-defined.

5) If the data type of either operand of an operator is approximate numeric, then the data type of the result is approximate numeric. The precision of the result is implementor-defined.

General Rules

1) If the value of any <primary> is the null value, then the result of the <value expression> is the null value.

2) If operators are not specified, then the result of the <value expression> is the value of the specified <primary>.

3) When a <value expression> is applied to a row of a table, each reference to a column of that table is a reference to the value of that column in that row.

4) The monadic arithmetic operators + and - specify monadic plus and monadic minus, respectively. Monadic plus does not change its operand. Monadic minus reverses the sign of its operand.

5) The dyadic arithmetic operators +, -, *, and / specify addition, subtraction, multiplication, and division, respectively. A divisor shall not be 0.

6) If the type of the result of an arithmetic operator is exact numeric, then:

   Case:

   a) If the operator is not division, then the mathematical result of the operation shall be exactly representable with the precision and scale of the result type.

   b) If the operator is division, then the approximate mathematical result of the operation represented with the precision and scale of the result type shall not lose any leading significant digits.

7) Expressions within parentheses are evaluated first and when the order of evaluation is not specified by parentheses, monadic operators are applied before multiplication and division, multiplication and division are applied before addition and subtraction, and operators at the same precedence level are applied from left to right.

5.10 <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 predicate>
            | <exists predicate>
```

Syntax Rules

None.

General Rules

1) The result of a <predicate> is derived by applying it to a given row of a table.

5.11 <comparison predicate>

Function

Specify a comparison of two values.

Format

<comparison predicate> ::=
<value expression> <comp op> {<value expression> | <subquery>}

<comp op> ::=
= | <> | < | > | <= | >=

Syntax Rules

1) The data types of the first <value expression> and the <subquery> or second <value expression> shall be comparable.

General Rules

1) Let x denote the result of the first <value expression> and let y denote the result of the <subquery> or the second <value expression>. The result of the <subquery> shall be at most one value.

2) If x or y is the null value or if the result of the <subquery> is empty, then "x <comp op> y" is unknown.

3) If x and y are nonnull values, then "x <comp op> y" is either true or false:

"x = y" is true if and only if x and y are equal.
"x <> y" is true if and only if x and y are not equal.
"x < y" is true if and only if x is less than y.
"x > y" is true if and only if x is greater than y.
"x <= y" is true if and only if x is not greater than y.
"x >= y" is true if and only if x is not less than y.

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

5) The comparison of two character strings is determined by the comparison of <character>s with the same ordinal position. If the strings do not have the same length, then the comparison is made with a working copy of the shorter string that has been effectively extended on the right with <space>s so that it has the same length as the other string.

6) Two strings are equal if all <character>s with the same ordinal position are equal. If two strings are not equal, then their relation is determined by the comparison of the first pair of unequal <character>s from the left end of the strings. This comparison is made with respect to the implementor-defined collating sequence.

7) Although "x = y" is unknown if both x and y are null values, in the contexts of GROUP BY, ORDER BY, and DISTINCT, a null value is identical to or is a duplicate of another null value.

5.12 <between predicate>

Function

Specify a range comparison.

Format

<between predicate> ::=
<value expression>
[NOT] BETWEEN <value expression> AND <value expression>

Syntax Rules

1) The data types of the three <value expression>s shall be comparable.

General Rules

1) Let x, y, and z denote the result of the first, second, and third <value expression>, respectively.

2) "x BETWEEN y AND z" has the same result as "x>=y AND x<=z".

3) "x NOT BETWEEN y AND z" has the same result as "NOT(x BETWEEN y AND z)".

5.13 <in predicate>

Function

Specify a quantified comparison.

Format

<in predicate> ::=
<value expression> [NOT] IN {<subquery> | (<in value list>)}

<in value list> ::=
<value specification>{,<value specification>}...

Syntax Rules

1) The data types of the first <value expression> and the <subquery> or all <value specification>s in the <in value list> shall be comparable.

General Rules

1) Let x denote the result of the <value expression>. Let S denote the result of the <subquery> as in a <quantified predicate>, or the values specified by the <in value list>.

2) "x IN S" has the same result as "x =ANY S". "x NOT IN S" has the same result as "NOT(x IN S)".

5.14 <like predicate>

Function

Specify a pattern-match comparison.

Format

<like predicate> ::=
<column specification> [NOT] LIKE <pattern>
[ESCAPE <escape character>]

<pattern> ::=
<value specification>

<escape character> ::=
<value specification>

Syntax Rules

1) The <column specification> shall reference a character string column.

2) The data type of the <pattern> shall be character string.

3) The data type of the <escape character> shall be character string of length 1.

General Rules

1) Let x denote the value referenced by the <column specification> and let y denote the result of the <value specification> of the <pattern>.

2) Case:

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

      i) Let z denote the result of the <value specification> of the <escape character>.

      ii) There shall be a partitioning of the string y into substrings such that each substring is of length 1 or 2, no substring of length 1 is the escape character z, and each substring of length 2 is the escape character z followed by either the escape character z, an underscore character, or the percent sign character. In that partitioning of y, each substring of length 2 represents a single occurrence of the second character of that substring. Each substring of length 1 that is the underscore character represents an arbitrary character specifier. Each substring of length 1 that is the percent sign character represents an arbitrary string specifier. Each substring of length 1 that is neither the underscore character nor the percent sign character represents the character that it contains.

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

3) The string y is a sequence of the minimum number of substring specifiers such that each <character> of y is part of exactly one substring specifier. A substring specifier is an arbitrary character specifier, an arbi-

trary string specifier, or any sequence of <character>s other than an arbitrary character specifier or an arbitrary string specifier.

4) "x LIKE y" is unknown if x or y is the null value. If x and y are nonnull values, then "x LIKE y" is either true or false.

5) "x LIKE y" is true if there exists a partitioning of x into substrings such that:

   a) A substring of x is a sequence of zero or more contiguous <character>s of x and each <character> of x is part of exactly one substring.

   b) If the i-th substring specifier of y is an arbitrary character specifier, the i-th substring of x is any single <character>.

   c) If the i-th substring specifier of y is an arbitrary string specifier, the i-th substring of x is any sequence of zero or more <character>s.

   d) If the i-th substring specifier of y is neither an arbitrary character specifier nor an arbitrary string specifier, the i-th substring of x is equal to that substring specifier and has the same length as that substring specifier.

   e) The number of substrings of x is equal to the number of substring specifiers of y.

6) "x NOT LIKE y" has the same result as "NOT(x LIKE y)".

5.15 <null predicate>

Function

Specify a test for a null value.

Format

<null predicate> ::=
<column specification> IS [NOT] NULL

Syntax Rules

None.

General Rules

1) Let x denote the value referenced by the <column specification>.

2) "x IS NULL" is either true or false.

3) "x IS NULL" is true if and only if x is the null value.

4) "x IS NOT NULL" has the same result as "NOT(x IS NULL)".

5.16 <quantified predicate>

Function

Specify a quantified comparison.

Format

<quantified predicate> ::=
<value expression>
<comp op> <quantifier> <subquery>

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

<all> ::= ALL

<some> ::= SOME | ANY

Syntax Rules

1) The data types of the <value expression> and the <subquery> shall be comparable.

General Rules

1) Let x denote the result of the <value expression> and let S denote the result of the <subquery>.

2) The result of "x <comp op> <quantifier> S" is derived by the application of the implied <comparison predicate> "x <comp op> s" to every value in S:

   Case:

   a) If S is empty or if the implied <comparison predicate> is true for every value s in S, then "x <comp op> <all> S" is true.

   b) If the implied <comparison predicate> is false for at least one value s in S, then "x <comp op> <all> S" is false.

   c) If the implied <comparison predicate> is true for at least one value s in S, then "x <comp op> <some> S" is true.

   d) If S is empty or if the implied <comparison predicate> is false for every value s in S, then "x <comp op> <some> S" is false.

   e) If "x <comp op> <quantifier> S" is neither true nor false, then it is unknown.

5.17 <exists predicate>

Function

Specify a test for an empty set.

Format

<exists predicate> ::=
EXISTS <subquery>

Syntax Rules

None.

General Rules

1) Let S denote the result of the <subquery>.

2) "EXISTS S" is either true or false.

3) "EXISTS S" is true if and only if S is not empty.

5.18 <search condition>

Function

Specify a condition that is "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 primary>

<boolean primary> ::=
<predicate> | (<search condition>)

Syntax Rules

1) A <column specification> or <value expression> specified in a <search condition> is directly contained in that <search condition> if the <column specification> or <value expression> is not specified within a <set function specification> or a <subquery> of that <search condition>.

General Rules

1) The result is derived by the application of the specified boolean operators to the conditions that result from the application of each specified <predicate> to a given row of a table or a given group of a grouped table. 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. AND and OR are defined by the following truth tables:

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

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

3) Expressions within parentheses are evaluated first and when the order of evaluation is not specified by parentheses, NOT is applied before AND, AND is applied before OR, and operators at the same precedence level are applied from left to right.

4) When a <search condition> is applied to a row of a table, each reference to a column of that table by a <column specification> directly contained in the <search condition> is a reference to the value of that column in that row.

5.19 <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) 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>.

General Rules

1) If all optional clauses are omitted, then the table is the result of the <from clause>. Otherwise, each specified clause is applied to the result of the previously specified clause and the table is the result of the application of the last specified clause. The result of a <table expression> is a derived table in which the i-th column inherits the description of the i-th column of the table specified by the <from clause>.

5.20 <from clause>

Function

Specify a table derived from one or more named tables.

Format

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

<table reference> ::=
<table name> [<correlation name>]

Syntax Rules

1) A <table name> specified in a <table reference> is exposed in the containing <from clause> if and only if that <table reference> does not specify a <correlation name>.

2) A <table name> that is exposed in a <from clause> shall not be the same as any other <table name> that is exposed in that <from clause>.

3) A <correlation name> specified in a <table reference> shall not be the same as any other <correlation name> specified in the containing <from clause>, and shall not be the same as the <table identifier> of any <table name> that is exposed in the containing <from clause>.

4) The scope of <correlation name>s and exposed <table name>s specified in a <from clause> is the innermost <subquery>, <query specification>, or <select statement> that contains the <table expression> in which the <from clause> is contained. A <table name> that is specified in a <from clause> has a scope defined by that <from clause> if and only if the <table name> is exposed in that <from clause>.

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

6) Case:

   a) If the <from clause> contains a single <table name>, then the description of the result of the <from clause> is the same as the description of the table identified by that <table name>.

   b) If the <from clause> contains more than one <table name>, then the description of the result of the <from clause> is the concatenation of the descriptions of the tables identified by those <table name>s, in the order in which the <table name>s appear in the <from clause>.

General Rules

1) The specification of a <correlation name> or exposed <table name> in a <table reference> defines that <correlation name> or <table name> as a designator of the table identified by the <table name> of that <table reference>.

2) Case:

a) If the <from clause> contains a single <table name>, then the result of the <from clause> is the table identified by that <table name>.

b) If the <from clause> contains more than one <table name>, then the result of the <from clause> is the extended Cartesian product of the tables identified by those <table name>s. The extended Cartesian product, R, is the multi-set 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 R is the product of the cardinalities of the identified tables. The ordinal position of a column in R is n+s, where n is the ordinal position of that column in the named 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>.

5.21 <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 denote the description of the result of the preceding <from clause>. Each <column specification> directly contained in the <search condition> shall unambiguously reference a column of T or be an outer reference.

   NOTE: "Outer reference" is defined in 5.7, "<column specification>" on page 26.

2) A <value expression> directly contained in the <search condition> shall not include a reference to a column derived from a function.

3) If a <value expression> directly contained in the <search condition> is a <set function specification>, then the <where clause> shall be contained in a <having clause> and the <column specification> in the <set function specification> shall be an outer reference.

   NOTE: "Outer reference" is defined in 5.7, "<column specification>" on page 26.

General Rules

1) Let R denote the result of the <from clause>.

2) The <search condition> is applied to each row of R. The result of the <where clause> is a table of those rows of R for which the result of the <search condition> is true.

3) Each <subquery> in the <search condition> is effectively executed for each row of R and the results used in the application of the <search condition> to the given row of R. If any executed <subquery> contains an outer reference to a column of R, then the reference is to the value of that column in the given row of R.

   NOTE: "Outer reference" is defined in 5.7, "<column specification>" on page 26.

5.22 <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 <column specification> [{, <column specification>}...]

Syntax Rules

1) Let T denote the description of the result of the preceding <from clause> or <where clause>.

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

General Rules

1) Let R denote the result of the preceding <from clause> or <where clause>.

2) The result of the <group by clause> is a partitioning of R 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, all values of that grouping column are identical.

3) Every row of a given group contains the same value 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.

5.23 <having clause>

Function

Specify a restriction on the grouped table resulting from the previous <group by clause> or <from clause> by eliminating groups not meeting the <search condition>.

Format

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

Syntax Rules

1) Let T denote the description of the result of the preceding <from clause>, <where clause>, or <group by clause>. Each <column specification> 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 5.7, "<column specification>" on page 26.

2) Each <column specification> 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>.

General Rules

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

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

3) When the <search condition> is applied to a given group of R, that group is the argument or argument source of each <set function specification> directly contained in the <search condition> unless the <column specification> in the <set function specification> is an outer reference.

   NOTE: "Outer reference" is defined in 5.7, "<column specification>" on page 26.

4) Each <subquery> in the <search condition> is effectively executed for each group of R and the result used in the application of the <search condition> to the given group of R. If any executed <subquery> contains an outer reference to a column of R, then the reference is to the values of that column in the given group of R.

   NOTE: "Outer reference" is defined in 5.7, "<column specification>" on page 26.

5.24 <subquery>

Function

Specify a multi-set of values derived from the result of a <table expression>.

Format

```
<subquery> ::=
          (SELECT [ALL | DISTINCT] <result specification> <table expression>)

<result specification> ::=
            <value expression>
          | *
```

Syntax Rules

1) The applicable privileges for each <table name> contained in the <table expression> shall include SELECT.

   NOTE: The "applicable <privileges>" for a <table name> are defined in 6.6, "<privilege definition>" on page 59.

2) Case:

   a) If the <result specification> "*" is specified in a <subquery> of any <predicate> other than an <exists predicate>, then the degree of the <table expression> shall be 1, and the <result specification> is equivalent to a <value expression> consisting of a <column specification> that references the sole column of the <table expression>.

   b) If the <result specification> "*" is specified in a <subquery> of an <exists predicate>, then the <result specification> is equivalent to an arbitrary <value expression> that does not include a <set function specification> and that is allowed in the <subquery>.

3) The data type of the values of the <subquery> is the data type of the implicit or explicit <value expression>.

4) Let R denote the result of the <table expression>.

5) Each <column specification> in the <value expression> shall unambiguously reference a column of R.

6) If R is a grouped view, then the <result specification> shall not contain a <set function specification>.

7) If R is a grouped table, then each <column specification> in the <value expression> shall reference a grouping column or be specified within a <set function specification>. If R is not a grouped table and the <value expression> includes a <set function specification>, then each <column specification> in the <value expression> shall be specified within a <set function specification>.

8) The <key word> DISTINCT shall not be specified more than once in a <subquery>, excluding any <subquery> contained in that <subquery>.

9) If a <subquery> is specified in a <comparison predicate>, then the <table expression> shall not contain a <group by clause> or a <having clause> and shall not identify a grouped view.

General Rules

1) If R is not a grouped table and the <value expression> includes a <set function specification>, then R is the argument or argument source of each <set function specification> in the <value expression> and the result of the <subquery> is the value specified by the <value expression>.

2) If R is not a grouped table and the <value expression> does not include a <set function specification>, then the <value expression> is applied to each row of R yielding a multi-set of n values, where n is the cardinality of R. If DISTINCT is not specified, then the multi-set is the result of the <subquery>. If DISTINCT is specified, then the result of the <subquery> is the set of values derived from that multi-set by the elimination of any redundant duplicate values.

3) If R is a grouped table, then the <value expression> is applied to each group of R yielding a multi-set of n values, where n is the number of groups in R. When the <value expression> is applied to a given group of R, that group is the argument or argument source of each <set function specification> in the <value expression>. If DISTINCT is not specified, then the multi-set is the result of the <subquery>. If DISTINCT is specified, then the result of the <subquery> is the set of values derived from that multi-set by the elimination of any redundant duplicate values.

5.25 <query specification>

Function

Specify a table derived from the result of a <table expression>.

Format

```
<query specification> ::=
        SELECT [ALL | DISTINCT] <select list> <table expression>

<select list> ::=
          <value expression> [{, <value expression>}...]
        | *
```

Syntax Rules

1) The applicable <privileges> for each <table name> contained in the <table expression> shall include SELECT.

   NOTE: The "applicable <privileges>" for a <table name> are defined in 6.6, "<privilege definition>" on page 59.

2) Let R denote the result of the <table expression>.

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

4) The <select list> "*" is equivalent to a <value expression> sequence in which each <value expression> is a <column specification> that references a column of R and each column of R is referenced exactly once. The columns are referenced in the ascending sequence of their ordinal position within R.

5) Each <column specification> in each <value expression> shall unambiguously reference a column of R. The <key word> DISTINCT shall not be specified more than once in a <query specification>, excluding any <subquery> of that <query specification>.

6) If R is a grouped view, then the <select list> shall not contain a <set function specification>.

7) If R is a grouped table, then each <column specification> in each <value expression> shall reference a grouping column or be specified within a <set function specification>. If R is not a grouped table and any <value expression> includes a <set function specification>, then every <column specification> in every <value expression> shall be specified within a <set function specification>.

8) Each column of the table that is the result of a <query specification> has the same data type, length, precision, and scale as the <value expression> from which the column was derived.

9) If the i-th <value expression> in the <select list> consists of a single <column specification>, then the i-th column of the result is a named column whose <column name> is that of the <column specification>. Otherwise, the i-th column is an unnamed column.

10) A column of the table that is the result of a <query specification> is constrained to contain only nonnull values if and only if it is a named column that is constrained to contain only nonnull values.

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

a) DISTINCT is not specified.

b) Every <value expression> in the <select list> consists of a <column specification>.

c) The <from clause> of the <table expression> specifies exactly one <table reference>, and that <table reference> refers to an updatable table.

d) The <where clause> of the <table expression> does not include a <subquery>.

e) The <table expression> does not include a <group by clause> or a <having clause>.

General Rules

1) If R is not a grouped table and the <select list> includes a <set function specification>, then R is the argument or argument source of each <set function specification> in the <select list> and the result of the <query specification> is a table consisting of one row. The i-th value of the row is the value specified by the i-th <value expression>.

2) If R is not a grouped table and the <select list> does not include a <set function specification>, then each <value expression> is applied to each row of R yielding a table of m rows, where m is the cardinality of R. The i-th column of the table contains the values derived by the applications of the i-th <value expression>. If DISTINCT is not specified, then the table is the result of the <query specification>. If 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.

3) If R is a grouped table that has zero groups and some <value expression> in the <select list> is a <column specification>, then the result of the <query specification> is an empty table. If R is a grouped table that has zero groups and every <value expression> in the <select list> is a <set function specification>, then the result of the <query specification> is a table having one row. The i-th value in that row is the result of the i-th <set function specification> in the <select list>.

4) If R is a grouped table that has one or more groups, then each <value expression> is applied to each group of R yielding a table of m rows, where m is the number of groups in R. The i-th column of the table contains the values derived by the applications of the i-th <value expression>. When a <value expression> is applied to a given group of R, that group is the argument or argument source of each <set function specification> in the <value expression>. If DISTINCT is not specified, then the table is the result of the <query specification>. If 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.

5) A row is a duplicate of another row if and only if all pairs of values with the same ordinal position are identical.

6. Schema definition language

6.1 <schema>

Function

Define a <schema>.

Format

<schema> ::=
CREATE SCHEMA <schema authorization clause>
[<schema element>...]

<schema authorization clause> ::=
AUTHORIZATION <schema authorization identifier>

<schema authorization identifier> ::=
<authorization identifier>

<schema element> ::=
<table definition>
| <view definition>
| <privilege definition>

Syntax Rules

1) The <schema authorization identifier> shall be different from the <schema authorization identifier> of any other <schema> in the same environment. The concept of environment is implementor-defined.

General Rules

None.

6.2 <table definition>

Function

Define a base table.

Format

<table definition> ::=
CREATE TABLE <table name>
(<table element> [{,<table element>}...])

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

Syntax Rules

1) If the <table name> contains an <authorization identifier>, then that <authorization identifier> shall be the same as the <schema authorization identifier> of the containing <schema>.

2) The <table name> shall be different from the <table name> of any other <table definition> or <view definition> in the containing <schema>.

3) The description of the table defined by a <table definition> includes the name <table name> and the column description specified by each <column definition>. The i-th column description is given by the i-th <column definition>.

General Rules

1) A <table definition> defines a base table.

6.3 <column definition>

Function

Define a column of a table.

Format

<column definition> ::=
<column name> <data type>
[NOT NULL [UNIQUE]]

Syntax Rules

1) The <column name> shall be different from the <column name> of any other <column definition> in the containing <table definition>.

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 of the column are specified by the <column name> and <data type>, respectively.

3) If NOT NULL is specified, then the column is constrained to contain only nonnull values.

4) Let C denote the <column name> of a <column definition>. If UNIQUE is specified, then the following <unique constraint definition> is implicit:

UNIQUE(C)

5) The description of the column defined by a <column definition> includes the name <column name> and the data type specified by the <data type>.

General Rules

1) If a column is constrained to contain only nonnull values, then the constraint is effectively checked after the execution of each <SQL statement>.

6.4 <unique constraint definition>

Function

Specify a uniqueness constraint for a table.

Format

<unique constraint definition> ::=
UNIQUE (<unique column list>)

<unique column list> ::=
<column name> [{,<column name>}...]

Syntax Rules

1) Let T denote the containing table.

2) 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.

3) The <column definition> for each <column name> in the <unique column list> shall specify NOT NULL.

General Rules

1) Let "designated columns" denote the columns identified by the <column name>s of the <unique column list>.

2) T is constrained to contain no rows that are duplicates with respect to the designated columns. Two rows are duplicates if the value of each designated column in the first row is equal to the value of the corresponding column in the second row. The constraint is effectively checked after the execution of each <SQL statement>.

6.5 <view definition>

Function

Define a viewed table.

Format

<view definition> ::=
CREATE VIEW <table name> [(<view column list>)]
AS <query specification>
[WITH CHECK OPTION]

<view column list> ::=
<column name> [{,<column name>}...]

Syntax Rules

1) If the <table name> contains an <authorization identifier>, then that <authorization identifier> shall be the same as the <schema authorization identifier> of the containing <schema>.

2) The <table name> shall be different from the <table name> of any other <view definition> or <table definition> in the containing <schema>.

3) If the <query specification> is updatable, then the viewed table is an updatable table. Otherwise, it is a read-only table.

4) If any two columns in the table specified by the <query specification> have the same <column name>, or if any column of that table is an unnamed column, then a <view column list> shall be specified.

5) The same <column name> shall not be specified more than once in the <view column list>.

6) The number of <column name>s in the <view column list> shall be the same as the degree of the table specified by the <query specification>.

7) The description of the table defined by a <view definition> includes the name <table name> and the column descriptions of the table specified by the <query specification>. If a <view column list> is specified, then the name of the i-th column is the i-th <column name> in that <view column list>.

8) If the <query specification> 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.

9) If WITH CHECK OPTION is specified, then the viewed table shall be updatable.

General Rules

1) A <view definition> defines a viewed table. The viewed table, V, is the table that would result if the <query specification> were executed. Whether a viewed table is materialized is implementor-defined.

2) If V is updatable, then let T denote the table identified by the <table name> specified in the first <from clause> in the <query specification>. For each row in V, there is a corresponding row in T from which the row of V is derived. For each column in V, there is a corresponding column in T from which the

column of V is derived. The insertion of a row into V is an insertion of a corresponding row into T. The deletion of a row from V is a deletion of the corresponding row in T. The updating of a column of a row in V is an updating of the corresponding row in T.

3) Case:

a) If WITH CHECK OPTION is specified and the <query specification> specifies a <where clause>, then an <insert statement>, an <update statement: positioned>, or an <update statement: searched> to the view shall not result in the creation of a row for which that <where clause> is false.

b) If WITH CHECK OPTION is not specified, then the <view definition> shall not constrain the data values that may be inserted into an updatable viewed table.

NOTE: See General Rule 2 of 8.7, "<insert statement>" on page 76, General Rule 5 of 8.11, "<update statement: positioned>" on page 83, and General Rule 4 of 8.12, "<update statement: searched>" on page 85.

6.6 <privilege definition>

Function

Define privileges.

Format

```
<privilege definition> ::=
        GRANT <privileges> ON <table name>
        TO <grantee> [{, <grantee>}...]
        [WITH GRANT OPTION]

<privileges> ::=
          ALL PRIVILEGES
        | <action> [{,<action>}...]

<action> ::=
          SELECT | INSERT | DELETE
        | UPDATE [(<grant column list>)]

<grant column list> ::=
        <column name> [{,<column name>}...]

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

Syntax Rules

1) Let T denote the table identified by the <table name>. The <privileges> specify one or more privileges on T.

2) UPDATE(<grant column list>) specifies the UPDATE privilege on each column of T identified in the <grant column list>. Each <column name> in the <grant column list> shall identify a column of T. If the <grant column list> is omitted, then UPDATE specifies the UPDATE privilege on all columns of T.

3) The applicable <privileges> for a reference to a <table name> are determined as follows:

   a) Case

      i) If the occurrence of the <table name> is contained in a <schema>, then let the applicable <authorization identifier> be the <authorization identifier> specified as the <schema authorization identifier> of the <schema>.

      ii) If the occurrence of the <table name> is contained in a <module>, then let the applicable <authorization identifier> be the <authorization identifier> specified as the <module authorization identifier> of the <module>.

   b) Case:

      i) If the applicable <authorization identifier> is the same as the <authorization identifier> explicitly or implicitly specified in the <table name>, then:

Case:

1. If T is a base table, then the applicable <privileges> are INSERT, SELECT, UPDATE, and DELETE, and those <privileges> are grantable.

2. If T is a viewed table that is not updatable, then the applicable <privileges> are SELECT, and that privilege is grantable if and only if the applicable SELECT privileges on all <table name>s contained in the <query specification> are grantable.

3. If T is a viewed table that is updatable, then the applicable <privileges> on T are the applicable <privileges> on the <table name> T2 specified in the <from clause> of the <query specification>. A privilege is grantable on T if and only if it is grantable on T2.

ii) If the applicable <authorization identifier> is not the same as the <authorization identifier> explicitly or implicitly specified in the <table name>, then the applicable <privilege definition>s consist of all <privilege definition>s whose <table name> is the same as the given <table name> and whose <grantee>s either contain the applicable <authorization identifier> or contain PUBLIC, and the applicable <privileges> consist of all <privileges> specified in applicable <privilege definition>s. A privilege is grantable if and only if it is specified in the <privileges> of some applicable <privilege definition> that specifies WITH GRANT OPTION and that specifies the applicable <authorization identifier>.

4) ALL is equivalent to a list of <action>s that include all applicable <privileges> on the <table name>.

5) The applicable <privileges> for the <table name> of a <privilege definition> shall include the <privileges> specified in the <privilege definition>.

General Rules

None.

7. Module language

7.1 <module>

Function

Define a module.

Format

<module> ::=
<module name clause>
<language clause>
<module authorization clause>
[<declare cursor>...]
<procedure>...

<language clause> ::=
LANGUAGE {COBOL | FORTRAN | PASCAL | PLI }

<module authorization clause> ::=
AUTHORIZATION <module authorization identifier>

<module authorization identifier> ::=
<authorization identifier>

Syntax Rules

1) For each <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>.

2) A <module> shall be associated with an application program during its execution. An application program shall be associated with at most one <module>.

General Rules

1) If the <language clause> of a <module> specifies COBOL (respectively FORTRAN, PASCAL, PLI) and if the agent that performs a call of a <procedure> in that <module> is not a standard COBOL program (respectively standard FORTRAN, Pascal, PL/I program), then the results are undefined.

2) After the last time that a programming language agent performs a call of a <procedure> in a <module>, a <commit statement> or <rollback statement> is implicitly performed. The choice of which of these <SQL statement>s to perform is implementor-defined. If an unrecoverable error has occurred, then the DBMS shall perform a <rollback statement>.

7.2 <module name clause>

Function

Name a <module>.

Format

<module name clause> ::=
MODULE [<module name>]

Syntax Rules

1) The <module name> shall be different from the <module name> of any other <module> in the same environment. The concept of environment is implementor-defined.

General Rules

1) A <module name clause> defines the optional <identifier> to be a <module name> that designates the containing <module> within the environment.

7.3 <procedure>

Function

Define a procedure and an SQL statement.

Format

```
<procedure> ::=
            PROCEDURE <procedure name> <parameter declaration>...;
            <SQL statement>;

<parameter declaration> ::=
              <parameter name> <data type>
            | <SQLCODE parameter>

<SQLCODE parameter> ::=
            SQLCODE

<SQL statement> ::=
              <close statement>
            | <commit statement>
            | <delete statement: positioned>
            | <delete statement: searched>
            | <fetch statement>
            | <insert statement>
            | <open statement>
            | <rollback statement>
            | <select statement>
            | <update statement: positioned>
            | <update statement: searched>
```

Syntax Rules

1) The <procedure name> shall be different from the <procedure name> of any other <procedure> in the containing <module>.

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 statement> of a <procedure> shall be specified in a <parameter declaration> in that <procedure>.

4) If a <column name> in an <SQL statement> is identical to a <parameter name> in the <parameter declaration> of the <procedure> containing the <SQL statement>, then the <column specification> that contains the <column name> shall contain a <qualifier>.

5) A valid call of a <procedure> shall supply n parameters, where n is the number of <parameter declaration>s in the <procedure>.

6) A <procedure> shall contain exactly one <SQLCODE parameter>. The parameter corresponding with the <SQLCODE parameter> is referred to as the SQLCODE parameter.

7) The subject <language clause> of a <procedure> is the <language clause> of the containing <module>.

8) Case:

a) If the subject <language clause> specifies COBOL, then:

i) The type of the SQLCODE parameter shall be COBOL usage COMPUTATIONAL picture S9(PC), where PC is an implementor-defined precision that is greater than or equal to 4.

ii) Any <data type> in a <parameter declaration> shall specify either CHARACTER or NUMERIC.

iii) If the i-th <parameter declaration> specifies a <data type> that is CHARACTER (L) for some <length> L, then the type of the i-th parameter shall be COBOL alphanumeric with a length of L.

iv) 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:

1. If S=P, then a PICTURE with an "S" followed by a "V" followed by P "9"s.

2. 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.

3. If S=0, then a PICTURE with an "S" followed by P "9"s followed by a "V".

b) If the subject <language clause> specifies FORTRAN, then:

i) The type of the SQLCODE parameter shall be FORTRAN INTEGER.

ii) Any <data type> in a <parameter declaration> shall specify either CHARACTER, INTEGER, REAL, or DOUBLE PRECISION.

iii) If the i-th <parameter declaration> specifies a <data type> that is CHARACTER(L) for some <length> L, then the type of the i-th parameter shall be FORTRAN CHARACTER with a length of L.

iv) 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.

c) If the subject <language clause> specifies PASCAL, then:

i) The type of the SQLCODE parameter shall be Pascal INTEGER.

ii) Any <data type> in a <parameter declaration> shall specify either CHARACTER, INTEGER, or REAL.

iii) If the i-th <parameter declaration> specifies a <data type> that is CHARACTER(L) for some <length> L, then the type of the i-th parameter shall be Pascal string with a length of L.

iv) 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 REAL.

d) If the subject <language clause> specifies PLI, then:

i) The type of the SQLCODE parameter shall be PL/I FIXED BINARY (PP), where PP is an implementor-defined precision that is greater than or equal to 15.

ii) Any <data type> in a <parameter declaration> shall specify either CHARACTER, DECIMAL, or FLOAT.

iii) If the i-th <parameter declaration> specifies a <data type> that is CHARACTER(L) for some <length> L, then the type of the i-th parameter shall be PL/I CHARACTER with a length of L.

iv) 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).

v) 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).

General Rules

1) A <procedure> defines a procedure that may be called by an implementor-defined agent.

2) When a <procedure> is called by a programming language agent:

a) If no transaction is active for that agent, then a transaction is effectively initiated and associated with this call and with subsequent calls by that agent of any <procedure> in the containing <module> until the agent terminates that transaction.

b) The <SQL statement> S of the <procedure> is executed.

3) Case:

a) If S executed successfully, then:

Case:

i) If S is a <fetch statement> for which a next row did not exist, then the SQLCODE parameter is set to 100.

ii) If S is an <insert statement> for which there was no candidate row, then the SQLCODE parameter is set to 100.

iii) If S is a <select statement> whose result was an empty table, then the SQLCODE parameter is set to 100.

iv) If S is a <update statement: searched> or <delete statement: searched> for which there were no object rows to update or delete, then the SQLCODE parameter is set to 100.

v) Otherwise, the SQLCODE parameter is set to 0.

b) If S did not execute successfully, then:

i) All changes made to the database by the execution of S are canceled.

ii) The SQLCODE parameter is set to a negative number whose value is implementor-defined.

8. Data manipulation language

8.1 <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>.

General Rules

1) Cursor CR shall be in the open state.

2) Cursor CR is placed in the closed state and the copy of the <cursor specification> of CR is destroyed.

8.2 <commit statement>

Function

Terminate the current transaction with commit.

Format

<commit statement> ::=
COMMIT WORK

Syntax Rules

None.

General Rules

1) The current transaction is terminated.

2) Any cursors that were opened by the current transaction are closed.

3) Any changes to the database that were made by the current transaction are committed.

8.3 <declare cursor>

Function

Define a cursor.

Format

```
<declare cursor> ::=
        DECLARE <cursor name> CURSOR
        FOR <cursor specification>

<cursor specification> ::=
        <query expression> [<order by clause>]

<query expression> ::=
          <query term>
        | <query expression> UNION [ALL] <query term>

<query term> ::=
        <query specification> | (<query expression>)

<order by clause> ::=
        ORDER BY <sort specification> [{, <sort specification>}...]

<sort specification> ::=
        {<unsigned integer> | <column specification>} [ASC | DESC]
```

Syntax Rules

1) The <cursor name> shall not be identical to the <cursor name> specified 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 Syntax Rule 1 of 7.1, "<module>" on page 61.

3) Let T denote the table specified by the <cursor specification>.

4) Case:

   a) If ORDER BY is specified, then T is a read-only table with the specified sort order.

   b) If neither ORDER BY nor UNION is specified and the <query specification> is updatable, then T is an updatable table.

   c) Otherwise, T is a read-only table.

5) Case:

a) If UNION is not specified, then the description of T is the description of the <query specification>.

b) If UNION is specified, then for each UNION that is specified, let T1 and T2 denote the tables specified by the <query expression> and the <query term>. The <select list>s for the specification of T1 and T2 shall consist of "*" or <column specification>s. Except for column names, the descriptions of T1 and T2 shall be identical. All columns of the result are unnamed. Except for <column name>s, the description of the result is the same as the description of T1 and T2.

6) If ORDER BY is specified, then each <column specification> in the <order by clause> shall identify a column of T, and each <unsigned integer> in the <order by clause> shall be greater than 0 and not greater than the degree of T. A named column may be referenced by an <unsigned integer> or a <column specification>. An unnamed column shall be referenced by an <unsigned integer>.

General Rules

1) Case:

a) If T is an updatable table, then the cursor is associated with the named table identified by the <table name> in the <from clause>. Let B denote that named table. For each row in T, there is a corresponding row in B from which the row of T is derived. When the cursor is positioned on a row of T, the cursor is also positioned on the corresponding row of B.

b) Otherwise, the cursor is not associated with a named table.

2) Case:

a) If UNION is not specified, then T is the result of the specified <query specification>.

b) If UNION is specified, then for each UNION that is specified let T1 and T2 be the result of the <query expression> and the <query term>. The result of the UNION is effectively derived as follows:

i) Initialize the result to an empty table.

ii) Insert each row of T1 and each row of T2 into the result.

iii) If ALL is not specified, then eliminate any redundant duplicate rows from the result.

3) Case:

a) If ORDER BY is not specified, then the ordering of rows in T is implementor-defined. This order is subject to the reproducibility requirement within a transaction, but it may change between transactions.

b) If ORDER BY is specified, then T has a sort order:

i) The sort order is a sequence of sort groups. A sort group is a sequence of rows in which all values of a sort column are identical. Furthermore, a sort group may be a sequence of sort groups.

ii) The cardinality of the sequence and the ordinal position of each sort group is determined by the values of the most significant sort column. The cardinality of the sequence is the minimum number of sort groups such that, for each sort group of more than one row, all values of that sort column are identical.

iii) If the sort order is based on additional sort columns, then each sort group of more than one row is a sequence of sort groups. The cardinality of each sequence and the ordinal position of each sort

group within each sequence is determined by the values of the next most significant sort column. The cardinality of each sequence is the minimum number of sort groups such that, for each sort group of more than one row, all values of that sort column are identical.

iv) The preceding paragraph applies in turn to each additional sort column. If a sort group consists of multiple rows and is not a sequence of sort groups, then the order of the rows within that sort group is undefined.

v) Let C be a sort column and let S denote a sequence which is determined by the values of C.

vi) A sort direction is associated with each sort column. If the direction of C is ascending, then the first sort group of S contains the lowest value of C and each successive sort group contains a value of C that is greater than the value of C in its predecessor sort group. If the direction is descending, then the first sort group of S contains the highest value of C and each successive sort group contains a value of C that is less than the value of C in its predecessor sort group.

vii) Ordering is determined by the comparison rules specified in 5.11, "<comparison predicate>" on page 32. The order of the null value relative to nonnull values is implementor-defined, but shall be either greater than or less than all nonnull values.

viii) A <sort specification> specifies a sort column and a direction. The sort column is the column referenced by the <unsigned integer> or the <column specification>. The <unsigned integer> n references the i-th column of R. A <column specification> references the named column.

ix) If DESC is specified in a <sort specification>, then the direction of the sort column specified by that <sort specification> is descending. If ASC is specified or if neither ASC nor DESC is specified, then the direction of the sort column is ascending.

x) The <sort specification> sequence determines the relative significance of the sort columns. The sort column specified by the first <sort specification> is the most significant sort column and each successively specified sort column is less significant than the previously specified sort column.

8.4 <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 applicable <privileges> for the <table name> shall include DELETE.

NOTE: The "applicable <privileges>" for a <table name> are defined in 6.6, "<privilege definition>" on page 59.

2) The containing <module> shall contain a <declare cursor> CR whose <cursor name> is the same as the <cursor name> in the <delete statement: positioned>.

3) The table designated by CR shall not be a read-only table.

4) Let T denote the table identified by the <table name>. T shall be the table identified in the first <from clause> in the <cursor specification> of CR.

General Rules

1) Cursor CR shall be positioned on a row.

2) The row from which the current row of CR is derived is deleted.

8.5 <delete statement: searched>

Function

Delete rows of a table.

Format

<delete statement: searched> ::=
DELETE FROM <table name>
[WHERE <search condition>]

Syntax Rules

1) The applicable <privileges> for the <table name> shall include DELETE.

   NOTE: The "applicable <privileges>" for a <table name> are defined in 6.6, "<privilege definition>" on page 59.

2) Let T denote the table identified by the <table name>. T shall not be a read-only table or a table that is identified in a <from clause> of any <subquery> contained in the <search condition>.

3) The scope of the <table name> is the entire <delete statement: searched>.

General Rules

1) Case:

   a) If a <search condition> is not specified, then all rows of T are deleted.

   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 all rows for which the result of the <search condition> is true are deleted. 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 5.7, "<column specification>" on page 26.

8.6 <fetch statement>

Function

Position a cursor on the next row of a table and retrieve values from that row.

Format

<fetch statement> ::=
FETCH <cursor name> INTO <fetch target list>

<fetch target list> ::=
<target specification> [{,<target specification>}...]

Syntax Rules

1) The containing <module> shall contain a <declare cursor> CR whose <cursor name> is the same as the <cursor name> of the <fetch statement>. Let T be the table defined by the <cursor specification> of CR.

2) The number of <target specification>s in the <fetch target list> shall be the same as the degree of table T.

3) Case:

   a) If the data type of the target designated by the i-th <target specification> in the <fetch target list> is character string, then the data type of the i-th column of table T shall be character string.

   b) If the data type of the target designated by the i-th <target specification> in the <fetch target list> is exact numeric, then the data type of the i-th column of table T shall be exact numeric.

   c) If the data type of the target designated by the i-th <target specification> in the <fetch target list> is approximate numeric, then the data type of the i-th column of table T shall be approximate numeric or exact numeric.

General Rules

1) Cursor CR shall be in the open state.

2) If the table designated by cursor CR is empty or if the position of CR is on or after the last row, CR is positioned after the last row, the value 100 is assigned to the SQLCODE parameter and values are not assigned to the targets identified by the <fetch target list>.

3) If the position of CR is before a row, CR is positioned on that row and values in that row are assigned to their corresponding targets.

4) If the position of CR is on r, where r is a row other than the last row, CR is positioned on the row immediately after r and values in the row immediately after r are assigned to their corresponding targets.

5) The assignment of values to targets in the <fetch target list> other than the SQLCODE parameter is in an implementor-defined order. The SQLCODE parameter is assigned a value last.

6) If an error occurs during the assignment of a value to a target, then the SQLCODE parameter is set to an implementor-defined negative number, and the values of targets other than the SQLCODE parameter are implementor-defined.

7) Let V be a target and let v denote its corresponding value in the current row of CR.

8) If v is the null value, then an indicator shall be specified for V, and that indicator is set to -1. If v is not the null value and V has an indicator, then:

   Case:

   a) If the data type of V is character string of length L and the length M of v is larger than L, then the indicator is set to M.

   b) Otherwise, the indicator is set to 0.

9) The target identified by the i-th <target specification> of the <fetch target list> corresponds to the i-th value in the current row of CR.

10) Case:

   a) If the data type of V is character string and the length of v is equal to the length of V, then the value of V is set to v.

   b) If the data type of V is character string of length L, and the length of v is larger than L, then the value of V is set to the first L characters of v.

   c) If the data type of V is character string of length L, and the length M of v is smaller than L, then the first M characters of V are set to v, and the last L-M characters of V are set to the space character.

   d) If the data type of V is exact numeric, then there shall be a representation of the value of v in the data type of V that does not lose any leading significant digits, and the value of V is set to that representation.

   e) If the data type of V is approximate numeric, then the value of V is set to the approximate value of v.

8.7 <insert statement>

Function

Create new rows in a table.

Format

<insert statement> ::=
INSERT INTO <table name> [(<insert column list>)]
{VALUES (<insert value list>) | <query specification>}

<insert column list> ::=
<column name> [{,<column name>}...]

<insert value list> ::=
<insert value> [{,<insert value>}...]

<insert value> ::=
<value specification> | NULL

Syntax Rules

1) The applicable <privileges> for the <table name> shall include INSERT.

NOTE: The "applicable <privileges>" for a <table name> are defined in 6.6, "<privilege definition>" on page 59.

2) Let T denote the table identified by the <table name>. T shall not be a read-only table or a table that is identified in a <from clause> of the <query specification> or of any <subquery> contained in the <query specification>.

3) Each <column name> in the <insert column list> shall identify a column of T and the same column shall not be identified more than once. Omission of the <insert column list> is an implicit specification of a <insert column list> that identifies all columns of T in the ascending sequence of their ordinal position within T.

4) A column identified by the <insert column list> is an object column.

5) Case:

   a) If an <insert value list> is specified, then the number of <insert value>s in that <insert value list> shall be equal to the number of <column name>s in the <insert column list>. Let the i-th item of the <insert statement> refer to the i-th <value specification> in that <insert value list>.

   b) If a <query specification> is specified, then the degree of the table specified by that <query specification> shall be equal to the number of <column name>s in the <insert column list>. Let the i-th item of the <insert statement> refer to the i-th column of the table specified by the <query specification>.

6) If the i-th item of the <insert statement> is not the <insert value> NULL, then:

   Case:

a) If the data type of the column of table T designated by the i-th <column name> is character string of length L, then the data type of the i-th item of the <insert statement> shall be character string of length less than or equal to L.

b) If the data type of the column of table T designated by the i-th <column name> is exact numeric, then the data type of the i-th item of the <insert statement> shall be exact numeric.

c) If the data type of the column of table T designated by the i-th <column name> is approximate numeric, then the data type of the i-th item of the <insert statement> shall be approximate numeric or exact numeric.

General Rules

1) A row is inserted in the following steps:

   a) A candidate row is effectively created in which the value of each column is the null value. If T is a base table, B, then the candidate row includes a null value for every column of B. If T is a viewed table, the candidate row includes a null value for every column of the base table, B, from which T is derived.

   b) For each object column in the candidate row, the value is replaced by an insert value.

   c) The candidate row is inserted in B.

2) If T is a viewed table defined by a <view definition> that specifies "WITH CHECK OPTION", then if the <query specification> contained in the <view definition> specifies a <where clause> that is not contained in a <subquery>, then the <search condition> of that <where clause> shall be true for the candidate row.

3) If an <insert value list> is specified, then:

   Case:

   a) If the i-th <insert value> of the <insert value list> is a <value specification>, then the value of the column of the candidate row corresponding with the i-th object column is the value of that <value specification>.

   b) If the i-th <insert value> of the <insert value list> is NULL, then the value of the column of the candidate row corresponding with the i-th object column is the null value.

4) If a <query specification> is specified, then let R be the result of the <query specification>. If R is empty, then the value 100 is assigned to the SQLCODE parameter and no row is inserted. The number of candidate rows created is equal to the cardinality of R. The insert values of one candidate row are the values in one row of R and the values in one row of R are the insert values of one candidate row.

5) Let V denote a row of R or the sequence of values specified by the <insert value list>. The i-th value of V is the insert value of the object column identified by the i-th <column name> in the <insert column list>.

6) Let C denote an object column. Let v denote a nonnull insert value of C.

7) Case:

   a) If the data type of C is character string and the length of v is equal to the length of C, then the value of C is set to v.

b) If the data type of C is character string of length L, and the length M of v is smaller than L, then the first M characters of C are set to v, and the last L-M characters of C are set to the space character.

c) If the data type of C is exact numeric, then there shall be a representation of the value of v in the data type of C that does not lose any leading significant digits, and the value of C is set to that representation.

d) If the data type of C is approximate numeric, then the value of C is set to the approximate value of v.

8.8 <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> of the <open statement>.

General Rules

1) Cursor CR shall be in the closed state.

2) Let S denote 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 each <target specification> is replaced by the value of the target that it identifies.

   b) If S specifies a read-only table, then that table, as specified by the copy of S, is effectively created.

   c) Cursor CR is placed in the open state and its position is before the first row of the table.

8.9 <rollback statement>

Function

Terminate the current transaction with rollback.

Format

<rollback statement> ::=
ROLLBACK WORK

Syntax Rules

None.

General Rules

1) Any changes to the database that were made by the current transaction are canceled.

2) Any cursors that were opened by the current transaction are closed.

3) The current transaction is terminated.

8.10 <select statement>

Function

Retrieve values from a specified row of a table.

Format

<select statement> ::=
SELECT [ALL | DISTINCT] <select list>
INTO <select target list>
<table expression>

<select target list> ::=
<target specification> [{,<target specification>}...]

Syntax Rules

1) The applicable <privileges> for each <table name> contained in the <table expression> shall include SELECT.

   NOTE: The "applicable <privileges>" for a <table name> are defined in 6.6, "<privilege definition>" on page 59.

2) The <table expression> shall not include a <group by clause> or a <having clause> and shall not identify a grouped view.

3) The number of elements in the <select list> shall be the same as the number of elements in the <select target list>.

4) Case:

   a) If the data type of the target designated by the i-th <target specification> in the <select target list> is character string, then the data type of the i-th <value expression> in the <select list> shall be character string.

   b) If the data type of the target designated by the i-th <target specification> in the <select target list> is exact numeric, then the data type of the i-th <value expression> in the <select list> shall be exact numeric.

   c) If the data type of the target designated by the i-th <target specification> in the <select target list> is approximate numeric, then the data type of the i-th <value expression> in the <select list> shall be approximate numeric or exact numeric.

General Rules

1) Let S be a <query specification> whose <select list> and <table expression> are those specified in the <select statement> and which specifies ALL or DISTINCT if it is specified in the <select statement>. Let R denote the result of <query specification> S.

2) The cardinality of R shall not be greater than one. If R is empty, then the value 100 is assigned to the SQLCODE parameter and values are not assigned to the targets identified by the <select target list>.

3) If R is not empty, then values in the row of R are assigned to their corresponding targets.

4) The assignment of values to targets in the <select target list> other than the SQLCODE parameter is in an implementor-defined order. The SQLCODE parameter is assigned a value last.

5) If an error occurs during the assignment of a value to a target, then the SQLCODE parameter is set to a negative number whose value is implementor-defined, and the values of targets other than the SQLCODE parameter are implementor-defined.

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 R.

7) Let V be an identified target and let v denote its corresponding value in the row of R.

8) If v is the null value, then an indicator shall be specified for V, and that indicator is set to -1. If v is not the null value and V has an indicator, then:

   Case:

   a) If the data type of V is character string of length L and the length M of v is larger than L, then the indicator is set to M.

   b) Otherwise, the indicator is set to 0.

9) Case:

   a) If the data type of V is character string and the length of v is equal to the length of V, then the value of V is set to v.

   b) If the data type of V is character string of length L, and the length of v is larger than L, then the value of V is set to the first L characters of v.

   c) If the data type of V is character string of length L, and the length M of v is smaller than L, then the first M characters of V are set to v, and the last L-M characters of V are set to the space character.

   d) If the data type of V is exact numeric, then there shall be a representation of the value of v in the data type of V that does not lose any leading significant digits, and the value of V is set to that representation.

   e) If the data type of V is approximate numeric, then the value of V is set to the approximate value of v.

8.11 <update statement: positioned>

Function

Update a row of a table.

Format

<update statement: positioned> ::=
UPDATE <table name>
SET <set clause : positioned> [{,<set clause : positioned>}...]
WHERE CURRENT OF <cursor name>

<set clause : positioned> ::=
<object column: positioned> = {<value expression> | NULL}

<object column: positioned> ::= <column name>

Syntax Rules

1) The applicable <privileges> for the <table name> shall include UPDATE for each <object column: positioned>.

NOTE: The "applicable <privileges>" for a <table name> are defined in 6.6, "<privilege definition>" on page 59.

2) The containing <module> shall contain a <declare cursor> CR whose <cursor name> is the same as the <cursor name> in the <update statement: positioned>.

3) The table designated by CR shall not be a read-only table.

4) Let T denote the table identified by the <table name>. T shall be the table identified in the first <from clause> in the <cursor specification> of CR.

5) A <value expression> in a <set clause : positioned> shall not include a <set function specification>.

6) Each <column name> specified as an <object column: positioned> shall identify a column of T. The same <object column: positioned> shall not appear more than once in an <update statement: positioned>.

7) The scope of the <table name> is the entire <update statement: positioned>.

8) For each <set clause : positioned>:

Case:

a) If NULL is specified, then the column designated by the <object column: positioned> shall allow nulls.

b) If the data type of the column designated by the <object column: positioned> is character string of length L, then the data type of the <value expression> shall be character string of length less than or equal to L.

c) If the data type of the column designated by the <object column: positioned> is exact numeric, then the data type of the <value expression> shall be exact numeric.

d) If the data type of the column designated by the <object column: positioned> is approximate numeric, then the data type of the <value expression> shall be approximate numeric or exact numeric.

General Rules

1) Cursor CR shall be positioned on a row.

2) The object row is that row from which the current row of CR is derived.

3) The object row is updated as specified by each <set clause : positioned>. A <set clause : positioned> specifies an object column and an update value of that column. The object column is the column identified by the <object column: positioned> in the <set clause : positioned>. The update value is the null value or the value specified by the <value expression>. If the <value expression> contains a reference to a column of T, the reference is to the value of that column in the object row before any value of the object row is updated.

4) The object row is updated in the following steps:

   a) A candidate row is created which is a copy of the object row.

   b) For each <set clause : positioned>, the value of the specified object column in the candidate row is replaced by the specified update value.

   c) The object row is replaced by the candidate row.

5) If T is a viewed table defined by a <view definition> that specifies "WITH CHECK OPTION", then if the <query specification> contained in the <view definition> specifies a <where clause> that is not contained in a <subquery>, then the <search condition> of that <where clause> shall be true for the candidate row.

6) Let C denote an object column. Let v denote a nonnull update value of C.

7) Case:

   a) If the data type of C is character string and the length of v is equal to the length of C, then the value of C is set to v.

   b) If the data type of C is character string of length L, and the length M of v is smaller than L, then the first M characters of C are set to v, and the last L-M characters of C are set to the space character.

   c) If the data type of C is exact numeric, then there shall be a representation of the value of v in the data type of C that does not lose any leading significant digits, and the value of C is set to that representation.

   d) If the data type of C is approximate numeric, then the value of C is set to the approximate value of v.

8.12 <update statement: searched>

Function

Update rows of a table.

Format

<update statement: searched> ::=
UPDATE <table name>
SET <set clause : searched> [{,<set clause : searched>}...]
[WHERE <search condition>]

<set clause : searched> ::=
<object column: searched> = {<value expression> | NULL}

<object column: searched> ::= <column name>

Syntax Rules

1) The applicable <privileges> for the <table name> shall include UPDATE for each <object column: searched>.

NOTE: The "applicable <privileges>" for a <table name> are defined in 6.6, "<privilege definition>" on page 59.

2) Let T denote the table identified by the <table name>. T shall not be a read-only table or a table that is identified in a <from clause> of any <subquery> that is contained in the <search condition>.

3) A <value expression> in a <set clause : searched> shall not include a <set function specification>.

4) Each <column name> specified as an <object column: searched> shall identify a column of T. The same <object column: searched> shall not appear more than once in an <update statement: searched>.

5) The scope of the <table name> is the entire <update statement: searched>.

6) For each <set clause : searched>:

Case:

a) If NULL is specified, then the column designated by the <object column: searched> shall allow nulls.

b) If the data type of the column designated by the <object column: searched> is character string of length L, then the data type of the <value expression> shall be character string of length less than or equal to L.

c) If the data type of the column designated by the <object column: searched> is exact numeric, then the data type of the <value expression> shall be exact numeric.

d) If the data type of the column designated by the <object column: searched> is approximate numeric, then the data type of the <value expression> shall be approximate numeric or exact numeric.

General Rules

1) 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. 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 5.7, "<column specification>" on page 26.

2) Each object row is updated as specified by each <set clause : searched>. A <set clause : searched> specifies an object column and an update value of that column. The object column is the column identified by the <object column: searched>. The update value is the null value or the value specified by the <value expression>. If the <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.

3) An object row is updated in the following steps:

   a) A candidate row is created which is a copy of the object row.

   b) For each <set clause : searched>, the value of the specified object column in the candidate row is replaced by the specified update value.

   c) The object row is replaced by the candidate row.

4) If T is a viewed table defined by a <view definition> that specifies "WITH CHECK OPTION", then if the <query specification> contained in the <view definition> specifies a <where clause> that is not contained in a <subquery>, then the <search condition> of that <where clause> shall be true for the candidate row.

5) Let C denote an object column. Let v denote a nonnull update value of C.

6) Case:

   a) If the data type of C is character string and the length of v is equal to the length of C, then the value of C is set to v.

   b) If the data type of C is character string of length L, and the length M of v is smaller than L, then the first M characters of C are set to v, and the last L-M characters of C are set to the space character.

   c) If the data type of C is exact numeric, then there shall be a representation of the value of v in the data type of C that does not lose any leading significant digits, and the value of C is set to that representation.

   d) If the data type of C is approximate numeric, then the value of C is set to the approximate value of v.

9. Levels

This standard specifies two levels. Level 2 is the complete SQL database language. Level 1 is the subset of Level 2 that obeys the following additional rules.

1) 4.16, "Transactions" on page 14:

   a) The first sentence of paragraph 1 is replaced with the following:

      A transaction is a sequence of operations, including database operations, that is atomic with respect to recovery.

   b) Paragraph 2 is deleted.

   c) The second sentence of paragraph 3 is deleted.

2) 5.3, "<token>" on page 18:

   An <identifier> shall not consist of more than 12 characters.

3) 5.4, "Names" on page 20:

   A <table name> shall not contain an <authorization identifier>.

4) 5.6, "<value specification> and <target specification>" on page 24:

   a) A <value specification> shall not contain USER.

   b) A <parameter specification> shall not specify an <indicator parameter>.

   c) A <variable specification> shall not specify an <indicator variable>.

5) 5.7, "<column specification>" on page 26:

   The following is added to Syntax Rule 4:

      A <column specification> shall not be an outer reference.

6) 5.8, "<set function specification>" on page 27, 5.24, "<subquery>" on page 48, and 5.25, "<query specification>" on page 50:

      An <all set function>, <subquery>, and <query specification> shall not contain ALL.

      NOTE: In Level 1, retention of duplicates is specified by omission of DISTINCT.

7) 5.8, "<set function specification>" on page 27:

      A <distinct set function> shall not contain AVG, MAX, MIN, or SUM.

8) 5.11, "<comparison predicate>" on page 32:

A <comp op> shall not contain "<>".

NOTE: In Level 1, a comparison of the form "A<>B" is expressed with the equivalent "NOT A=B".

9) 5.14, "<like predicate>" on page 35:

a) A <like predicate> shall not specify ESCAPE <escape character>.

b) A <like predicate> shall not specify NOT.

NOTE: In Level 1, a <like predicate> containing NOT would be expressed with the equivalent <search condition> of "NOT <like predicate>".

10) 5.17, "<exists predicate>" on page 39:

A <predicate> shall not specify an <exists predicate>.

11) 5.22, "<group by clause>" on page 46:

The following sentence is added to General Rule 2:

The grouping of rows in which the value of one or more grouping column is null is implementor-defined.

12) 5.25, "<query specification>" on page 50:

Syntax Rule 11 is replaced with the following:

The determination of whether a <query specification> is updatable or read-only is implementor-defined.

13) 6.1, "<schema>" on page 53:

A <schema> shall not be specified. A Level 1 implementation shall provide some mechanism to associate an <authorization identifier> with a <table definition>, <view definition>, or <privilege definition>.

14) 6.2, "<table definition>" on page 54:

A <table definition> shall not contain a <unique constraint definition>. A Level 1 implementation shall provide some mechanism for specifying a uniqueness constraint for a table.

15) 6.3, "<column definition>" on page 55:

a) A <data type> in a <column definition> shall not contain REAL, DOUBLE PRECISION, or NUMERIC.

b) A <column definition> shall specify NOT NULL.

c) A <column definition> shall not specify UNIQUE.

16) 6.5, "<view definition>" on page 57:

A <view definition> shall not contain WITH CHECK OPTION.

17) 6.6, "<privilege definition>" on page 59:

A <privilege definition> shall not contain WITH GRANT OPTION.

18) 7.3, "<procedure>" on page 63:

a) Syntax Rule 8 Case (a) (2) is replaced with the following:

Any <data type> in a <parameter declaration> shall specify CHARACTER.

b) In General Rule 3 Case (a), each occurrence of the number "100" is replaced with "a positive number whose value is implementor-defined".

c) General Rule 3 Case (b) (1) is replaced with the following:

Whether changes made to the database by the execution of S are canceled is implementor-defined.

19) 8.3, "<declare cursor>" on page 69:

a) A <sort specification> shall not contain an <unsigned integer>.

b) A <sort specification> shall not contain ASC.

NOTE: In Level 1, ascending order is specified by omission of DESC.

c) A <query expression> shall not contain UNION.

NOTE: In Level 1, the union function is not provided.

20) 8.7, "<insert statement>" on page 76:

An <insert statement> shall not contain a <query specification>.

21) 8.11, "<update statement: positioned>" on page 83 and 8.4, "<delete statement: positioned>" on page 72:

An <SQL statement> shall not specify an <update statement: positioned> or a <delete statement: positioned>.

Annexes

A. <embedded SQL host program>

(This annex is not an integral part of the body of the standard.)

Function

Specify an embedded SQL application program.

Format

```
<embedded SQL host program> ::=
        <embedded SQL COBOL program>
      | <embedded SQL FORTRAN program>
      | <embedded SQL Pascal program>
      | <embedded SQL PL/I program>

<embedded SQL statement> ::=
      <SQL prefix>
        { <declare cursor>
        | <embedded exception declaration>
        | <SQL statement>}
      [<SQL terminator>]

<SQL prefix> ::=
      EXEC SQL

<SQL terminator> ::=
      END-EXEC | ;

<embedded SQL declare section> ::=
      <embedded SQL begin declare>
      [<host variable definition>...]
      <embedded SQL end declare>

<embedded SQL begin declare> ::=
      <SQL prefix> BEGIN DECLARE SECTION [<SQL terminator>]

<embedded SQL end declare> ::=
      <SQL prefix> END DECLARE SECTION [<SQL terminator>]
```

<host variable definition> ::=
<COBOL variable definition>
| <FORTRAN variable definition>
| <Pascal variable definition>
| <PL/I variable definition>

<embedded variable name> ::=
:<host identifier>

<host identifier> ::=
<COBOL host identifier>
| <FORTRAN host identifier>
| <Pascal host identifier>
| <PL/I host identifier>

Syntax Rules

1) An <embedded SQL host program> is an application program 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.

2) 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. 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>. An <embedded SQL statement>, <embedded SQL begin declare>, or <embedded SQL end declare> contained in an <embedded SQL PL/I program> shall contain an <SQL terminator> that is a semi-colon. In an <embedded SQL Pascal program>, an <embedded SQL begin declare> shall contain an <SQL terminator> that is a semi-colon; an <embedded SQL end declare> or an <embedded SQL statement> that is immediately followed by an <embedded SQL begin declare> or an <embedded SQL statement> shall contain an <SQL terminator> that is a semi-colon; otherwise, an <embedded SQL end declare> or <embedded SQL statement> shall not contain an <SQL terminator> but shall be terminated according to the rules for Pascal statements.

3) An <SQL prefix>, <embedded SQL begin declare>, or <embedded SQL end declare> shall be specified within one line with no comments. Otherwise, the rules for the continuation of lines and tokens from one line to the next and for the placement of comments are those of the programming language of the containing <embedded SQL host program>.

4) 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>.

5) Any <host identifier> that occurs 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>. 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>.

6) A <host variable definition> defines the host language data type of the <host identifier>. For each such host language data type an equivalent SQL data type is specified in annex C, "<embedded SQL COBOL program>" on page 97, annex D, "<embedded SQL FORTRAN program>" on page 99, annex

E, "<embedded SQL Pascal program>" on page 101, and annex F, "<embedded SQL PL/I program>" on page 103.

7) Given an <embedded SQL host program> H, there is an implied SQL <module> M derived from H as follows:

   a) M contains a <module name clause> whose <module name> is either implementor-defined or is omitted.

   b) M contains a <language clause> that specifies either COBOL, FORTRAN, PASCAL, or PLI, where H is respectively an <embedded SQL COBOL program>, an <embedded SQL FORTRAN program>, an <embedded SQL Pascal program>, or an <embedded SQL PL/I program>.

   c) M contains a <module authorization clause> whose <module authorization identifier> is implementor-defined.

   d) M contains one <declare cursor> for each <declare cursor> contained in H. Each <declare cursor> of M is a copy of the corresponding <declare cursor> of H in which each distinct <embedded variable name> has been consistently replaced with a distinct implementor-defined <parameter name>.

   e) M contains a <procedure> for each <SQL statement> contained in H. The <procedure> PS of M corresponding with an <SQL statement> ES of H is defined as follows:

      i) The <procedure name> of PS is implementor-defined.

      ii) The <SQL statement> of PS is a copy of ES in which each distinct <embedded variable name> has been consistently replaced with a distinct implementor-defined <parameter name>.

      iii) PS contains a <parameter declaration> for each distinct implementor-defined <parameter name> contained in the <SQL statement> of PS, and a <parameter declaration> that specifies SQLCODE. The order of the <parameter declarations>s is implementor-defined. The <parameter declaration> corresponding with a given distinct <embedded variable name> V that occurs in ES specifies the distinct <parameter name> with which V was replaced, and the SQL <data type> that is equivalent to the host language data type of V.

8) Given an <embedded SQL host program> H, there is an implied standard conforming host program P derived from H as follows:

   a) Each <embedded SQL begin declare> and each <embedded SQL end declare> has been deleted.

   b) Each <embedded SQL statement> that contains a <declare cursor> or an <embedded exception declaration> has been deleted.

   c) Each <embedded SQL statement> that contains an <SQL 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 statement>. The arguments of the CALL include each distinct <host identifier> contained in the <SQL statement> together with the SQLCODE variable. The order of the arguments in the CALL corresponds with the order of the corresponding <parameter declaration>s in the corresponding <procedure>.

      NOTE: The "SQLCODE variable" is abbreviated to "SQLCOD" in an <embedded SQL FORTRAN program>. See Syntax Rule 6 of annex D, "<embedded SQL FORTRAN program>" on page 99.

ii) Exception actions, as specified in annex B, "<embedded exception declaration>" on page 95.

9) 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.

10) 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 SQL <module> as specified by the Formats and Syntax Rules of this standard.

b) If H is an <embedded SQL COBOL program>, then the implied program P shall be a standard conforming COBOL program. If H is an <embedded SQL FORTRAN program>, then the implied program P shall be a standard conforming FORTRAN program. If H is an <embedded SQL Pascal program>, then the implied program P shall be a standard conforming Pascal program. If H is an <embedded SQL PL/I program>, then the implied program P shall be a standard conforming PL/I program.

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.

B. <embedded exception declaration>

(This annex is not an integral part of the body of the standard.)

Function

Specify the action to be taken when an <SQL statement> causes an exception condition.

Format

<embedded exception declaration> ::=
WHENEVER <condition> <exception action>

<condition> ::=
SQLERROR | NOT FOUND

<exception action> ::=
CONTINUE | <go to>

<go to> ::=
{GOTO | GO TO} <target>

<target> ::= :<host identifier> | <unsigned integer>

Syntax Rules

1) Case:

   a) If an <embedded exception declaration> is contained in an <embedded SQL COBOL program>, then the <target> of a <go to> shall specify a <host identifier> that is a section-name or an unqualified paragraph-name.

   b) If an <embedded exception declaration> is contained in an <embedded SQL FORTRAN program>, then the <target> of a <go to> shall be an <unsigned integer> that is the statement label of an executable statement that appears in the same program unit as the <go to>.

   c) If an <embedded exception declaration> is contained in an <embedded SQL Pascal program>, then the <target> of a <go to> shall be an <unsigned integer> that is a label.

   d) If an <embedded exception declaration> is contained in an <embedded SQL PL/I program>, then the <target> of a <go to> shall specify a <host identifier> that is a label constant or a label variable.

2) An <embedded exception declaration> contained in an <embedded SQL host program> applies to an <SQL statement> contained in that <embedded SQL host program> if and only if the <SQL statement> occurs 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> occurs between the <embedded exception declaration> and the <SQL statement> in the text sequence of the <embedded SQL host program>.

3) If an <embedded exception declaration> specifies a <go to>, then the <host identifier> or <unsigned integer> of the <go to> shall be such that a host language GO TO statement specifying that <host identi-

fier> or <unsigned integer> is valid at every <SQL statement> to which the <embedded exception declaration> applies.

General Rules

1) Immediately after the execution of an <SQL statement> in an <embedded SQL host program>:

Case:

a) If the value of the SQLCODE (SQLCOD) variable is +100 and the <embedded SQL host program> contains an <embedded exception declaration> that applies to the <SQL statement> and whose <condition> is NOT FOUND and whose <exception action> is a <go to>, then a GO TO statement of the host language is performed, specifying the <host identifier> or <unsigned integer> of the <go to>.

b) If the value of the SQLCODE (SQLCOD) variable is a negative number and the <embedded SQL host program> contains an <embedded exception declaration> that applies to the <SQL statement> and whose <condition> is SQLERROR and <exception action> is a <go to>, then a GO TO statement of the host language is performed, specifying the <host identifier> or <unsigned integer> of the <go to>.

c) If the <embedded SQL host program> contains no <embedded exception declaration> that applies to the <SQL statement>, or if it contains an <embedded exception declaration> that applies to the <SQL statement> and that specifies CONTINUE, then no further action is performed for the <SQL statement>.

C. <embedded SQL COBOL program>

(This annex is not an integral part of the body of the standard.)

Function

Specify an SQL module embedded in a COBOL program.

Format

```
<embedded SQL COBOL program> ::=
        See the Syntax Rules.

<COBOL variable definition> ::=
        {01 | 77} <COBOL host identifier> <COBOL type specification>
        [<character>...] .

<COBOL host identifier> ::=
        See Syntax Rule 3.

<COBOL type specification> ::=
            <COBOL character type>
          | <COBOL numeric type>
          | <COBOL integer type>

<COBOL character type> ::=
        PIC[TURE] [IS] X(<integer>)

<COBOL numeric type> ::=
        PIC[TURE] [IS]
        S{<nines> [V<nines>]
           | <nines>V
           | V<nines>}
        [USAGE [IS]] DISPLAY SIGN LEADING SEPARATE

<COBOL integer type> ::=
        PIC[TURE] [IS]
        S<nines>
        [USAGE [IS]] COMP[UTATIONAL]

<nines> ::= {9[(<integer>)]}...
```

Syntax Rules

1) An <embedded SQL COBOL program> is an application program that consists of COBOL text and SQL text. The COBOL text shall conform to standard COBOL. 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 variable 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.

   a) A <COBOL variable definition> shall be a valid data description entry in the Data Division of the program derived from the <embedded SQL COBOL program>.

   b) The optional <character> sequence in a <COBOL variable definition> may specify a VALUE clause. Whether other clauses may be specified is implementor-defined. The <character> sequence shall be such that the <COBOL variable definition> is a valid COBOL data description entry.

   c) A <COBOL character type> describes a character string variable. The equivalent SQL data type is CHARACTER of the same length.

   d) A <COBOL numeric type> describes an exact numeric variable. The equivalent SQL data type is NUMERIC of the same precision and scale.

   e) A <COBOL integer type> describes an exact numeric variable. The equivalent SQL data type is INTEGER.

5) An <embedded SQL COBOL program> shall contain a variable named SQLCODE defined with a data type of usage COMPUTATIONAL picture S9(PC), where PC is the implementor-defined precision specified for SQLCODE parameters in 7.3, "<procedure>" on page 63.

General Rules

See annex A, "<embedded SQL host program>" on page 91.

D. <embedded SQL FORTRAN program>

(This annex is not an integral part of the body of the standard.)

Function

Specify an SQL module embedded in a FORTRAN program.

Format

<embedded SQL FORTRAN program> ::=
See the Syntax Rules.

<FORTRAN variable definition> ::=
<FORTRAN type specification>
<FORTRAN host identifier> [{,<FORTRAN host identifier>}...]

<FORTRAN host identifier> ::=
See Syntax Rule 4.

<FORTRAN type specification> ::=
CHARACTER [*<length>]
| INTEGER
| REAL
| DOUBLE PRECISION

Syntax Rules

1) An <embedded SQL FORTRAN program> is an application program that consists of FORTRAN text and SQL text. The FORTRAN text shall conform to standard FORTRAN. 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 5.3, "<token>" on page 18.

4) A <FORTRAN host identifier> is any valid FORTRAN variable name. 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 a host variable.

   a) A <FORTRAN variable definition> shall be a valid FORTRAN type-statement in the program derived from the <embedded SQL FORTRAN program>.

b) CHARACTER describes a character string variable. The equivalent SQL data type is CHARACTER of the same length.

c) INTEGER describes an exact numeric variable. The equivalent SQL data type is INTEGER.

d) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

e) DOUBLE PRECISION describes an approximate numeric variable. The equivalent SQL data type is DOUBLE PRECISION.

6) An <embedded SQL FORTRAN program> shall contain a variable named SQLCOD defined with a data type of INTEGER. In an <embedded SQL FORTRAN program>, SQLCOD shall be used as the abbreviation for SQLCODE.

General Rules

See annex A, "<embedded SQL host program>" on page 91.

E. <embedded SQL Pascal program>

(This annex is not an integral part of the body of the standard.)

Function

Specify an SQL module embedded in a Pascal program.

Format

<embedded SQL Pascal program> ::=
See the Syntax Rules.

<Pascal variable definition> ::=
<Pascal host identifier> [{,<Pascal host identifier>}...]:
<Pascal type specification>

<Pascal host identifier> ::=
See Syntax Rule 3.

<Pascal type specification> ::=
PACKED ARRAY [1..<length>] OF CHAR
| INTEGER
| REAL

Syntax Rules

1) An <embedded SQL Pascal program> is an application program that consists of Pascal text and SQL text. The Pascal text shall conform to standard Pascal. 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 any valid Pascal variable-identifier. A <Pascal host identifier> shall be contained in an <embedded SQL Pascal program>.

4) A <Pascal variable definition> defines a host variable.

   a) A <Pascal variable definition> shall be a valid Pascal variable-declaration in the program derived from the <embedded SQL Pascal program>.

   b) PACKED ARRAY [1..<length>] OF CHAR describes a character string variable. The equivalent SQL data type is CHARACTER with the same length.

   c) INTEGER describes an exact numeric variable. The equivalent SQL data type is INTEGER.

   d) REAL describes an approximate numeric variable. The equivalent SQL data type is REAL.

5) An <embedded SQL Pascal program> shall contain a variable named SQLCODE defined with a data type of INTEGER.

General Rules

See annex A, "<embedded SQL host program>" on page 91.

F. <embedded SQL PL/I program>

(This annex is not an integral part of the body of the standard.)

Function

Specify an SQL module embedded in a PL/I program.

Format

```
<embedded SQL PL/I program> ::=
        See the Syntax Rules.

<PL/I variable definition> ::=
        {DCL | DECLARE}
          {<PL/I host identifier>
            | (<PL/I host identifier> [{,<PL/I host identifier>}...])}
        <PL/I type specification>
        [<character>...];

<PL/I host identifier> ::=
        See Syntax Rule 3.

<PL/I type specification> ::=
            CHAR[ACTER] (<length>)
          | {DEC[IMAL] FIXED | FIXED DEC[IMAL]} (<precision> [,<scale>] )
          | {BIN[ARY] FIXED | FIXED BIN[ARY]} [(<precision>)]
          | {BIN[ARY] FLOAT | FLOAT BIN[ARY]} (<precision>)
```

Syntax Rules

1) An <embedded SQL PL/I program> is an application program that consists of PL/I text and SQL text. The PL/I text shall conform to standard PL/I. 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.

   a) A <PL/I variable definition> shall be a valid PL/I data declaration in the program derived from the <embedded SQL PL/I program>.

   b) A <PL/I variable definition> shall specify a scalar variable, not an array or structure.

c) The optional <character> sequence in a <PL/I variable definition> may specify an INITIAL clause. Whether other clauses may be specified is implementor-defined. The <character> sequence shall be such that the <variable definition> is a valid PL/I DECLARE statement.

d) CHAR[ACTER] describes a character string variable. The equivalent SQL data type is CHARACTER with the same length.

e) FIXED DEC[IMAL] 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>.

f) FLOAT BIN[ARY] describes an approximate numeric variable. The equivalent SQL data type is FLOAT with the same <precision>.

g) FIXED BIN[ARY] describes an exact numeric variable. The equivalent SQL data type is INTEGER.

5) An <embedded SQL PL/I program> shall contain a variable named SQLCODE defined with a data type of FIXED BINARY(PP), where PP is the implementor-defined <precision> specified for SQLCODE parameters in 7.3, "<procedure>" on page 63.

General Rules

See annex A, "<embedded SQL host program>" on page 91.

Index

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X3.115-1984 Unformatted 80 Megabyte Trident Pack for Use at 370 tpi and 6000 bpi (General, Physical, and Magnetic Characteristics)
X3.116-1986 Recorded Magnetic Tape Cartridge, 4-Track, Serial 0.250 Inch (6.30 mm) 6400 bpi (252 bpmm), Inverted Modified Frequency Modulation Encoded
X3.117-1984 Printable/Image Areas for Text and Facsimile Communication Equipment
X3.118-1984 Financial Services — Personal Identification Number — PIN Pad
X3.119-1984 Contact Start/Stop Storage Disk, 158361 Flux Transitions per Track, 8.268 Inch (210 mm) Outer Diameter and 3.937 inch (100 mm) Inner Diameter
X3.120-1984 Contact Start/Stop Storage Disk
X3.121-1984 Two-Sided, Unformatted, 8-Inch (200-mm), 48-tpi, Double-Density, Flexible Disk Cartridge for 13 262 ftpr Two-Headed Application
X3.122-1986 Computer Graphics Metafile for the Storage and Transfer of Picture Description Information
X3.124-1985 Graphical Kernel System (GKS) Functional Description
X3.124.1-1985 Graphical Kernel System (GKS) FORTRAN Binding
X3.125-1985 Two-Sided, Double-Density, Unformatted 5.25-inch (130-mm), 48-tpi (1,9-tpmm), Flexible Disk Cartridge for 7958 bpr Use
X3.126-1986 One- or Two-Sided Double-Density Unformatted 5.25-inch (130-mm), 96 Tracks per Inch, Flexible Disk Cartridge
X3.127-1987 Unrecorded Magnetic Tape Cartridge for Information Interchange
X3.128-1986 Contact Start-Stop Storage Disk — 83 000 Flux Transitions per Track, 130-mm (5.118-in) Outer Diameter and 40-mm (1.575-in) Inner Diameter
X3.129-1986 Intelligent Peripheral Interface, Physical Level
X3.130-1986 Intelligent Peripheral Interface, Logical Device Specific Command Sets for Magnetic Disk Drive
X3.131-1986 Small Computer Systems Interface
X3.132-1987 Intelligent Peripheral Interface — Logical Device Generic Command Set for Optical and Magnetic Disks
X3.133-1986 Database Language —NDL
X3.135-1986 Database Language — SQL
X3.136-1986 Serial Recorded Magnetic Tape Cartridge for Information Interchange, Four and Nine Track
X3.139-1987 Fiber Distributed Data Interface (FDDI) Token Ring Media Access Control (MAC)
X3.140-1986 Open Systems Interconnection — Connection Oriented Transport Layer Protocol Specification
X3.141-1987 Data Communication Systems and Services — Measurement Methods for User-Oriented Performance Evaluation
X3.146-1987 Device Level Interface for Streaming Cartridge and Cassette Tape Drives
X3.147-1987 Intelligent Peripheral Interface — Logical Device Generic Command Set for Magnetic Tapes
X3.153-1987 Open Systems Interconnection — Basic Connection Oriented Session Protocol Specification
X11.1-1977 Programming Language MUMPS
IEEE 416-1978 Abbreviated Test Language for All Systems (ATLAS)
IEEE 716-1982 Standard C/ATLAS Language
IEEE 717-1982 Standard C/ATLAS Syntax
IEEE 770X3.97-1983 Programming Language PASCAL
IEEE 771-1980 Guide to the Use of ATLAS
ISO 8211-1986 Specifications for a Data Descriptive File for Information Interchange
MIL-STD-1815A-1983 Reference Manual for the Ada Programming Language
NBS-ICST 1-1986 Fingerprint Identification — Data Format for Information Interchange

---

X3/TRI-82 Dictionary for Information Processing Systems (Technical Report)

American National Standards for Information Processing

X3.1-1976 Synchronous Signaling Rates for Data Transmission
X3.2-1970 Print Specifications for Magnetic Ink Character Recognition
X3.4-1986 Coded Character Sets — 7-Bit ASCII
X3.5-1970 Flowchart Symbols and Their Usage
X3.6-1965 Perforated Tape Code
X3.9-1978 Programming Language FORTRAN
X3.11-1969 General Purpose Paper Cards
X3.14-1983 Recorded Magnetic Tape (200 CPI, NRZI)
X3.15-1976 Bit Sequencing of the American National Standard Code for Information Interchange in Serial-by-Bit Data Transmission
X3.16-1976 Character Structure and Character Parity Sense for Serial-by-Bit Data Communication in the American National Standard Code for Information Interchange
X3.17-1981 Character Set for Optical Character Recognition (OCR-A)
X3.18-1974 One-Inch Perforated Paper Tape
X3.19-1974 Eleven-Sixteenths-Inch Perforated Paper Tape
X3.20-1967 Take-Up Reels for One-Inch Perforated Tape
X3.21-1967 Rectangular Holes in Twelve-Row Punched Cards
X3.22-1983 Recorded Magnetic Tape (800 CPI, NRZI)
X3.23-1985 Programming Language COBOL
X3.25-1976 Character Structure and Character Parity Sense for Parallel-by-Bit Data Communication in the American National Standard Code for Information Interchange
X3.26-1980 Hollerith Punched Card Code
X3.27-1978 Magnetic Tape Labels and File Structure
X3.28-1976 Procedures for the Use of the Communication Control Characters of American National Standard Code for Information Interchange in Specified Data Communication Links
X3.29-1971 Specifications for Properties of Unpunched Oiled Paper Perforator Tape
X3.30-1986 Representation for Calendar Date and Ordinal Date
X3.31-1973 Structure for the Identification of the Counties of the United States
X3.32-1973 Graphic Representation of the Control Characters of American National Standard Code for Information Interchange
X3.34-1972 Interchange Rolls of Perforated Tape
X3.37-1980 Programming Language APT
X3.38-1972 Identification of States of the United States (Including the District of Columbia)
X3.39-1986 Recorded Magnetic Tape (1600 CPI, PE)
X3.40-1983 Unrecorded Magnetic Tape (9-Track 800 CPI, NRZI; 1600 CPI, PE; and 6250 CPI, GCR)
X3.41-1974 Code Extension Techniques for Use with the 7-Bit Coded Character Set of American National Standard Code for Information Interchange
X3.42-1975 Representation of Numeric Values in Character Strings
X3.43-1986 Representations of Local Time of Day
X3.44-1974 Determination of the Performance of Data Communication Systems
X3.45-1982 Character Set for Handprinting
X3.46-1974 Unrecorded Magnetic Six-Disk Pack (General, Physical, and Magnetic Characteristics)
X3.47-1977 Structure for the Identification of Named Populated Places and Related Entities of the States of the United States for Information Interchange
X3.48-1986 Magnetic Tape Cassettes (3.81-mm [0.150-Inch] Tape at 32 bpmm [800 bpi], PE)
X3.49-1975 Character Set for Optical Character Recognition (OCR-B)
X3.50-1986 Representations for U.S. Customary, SI, and Other Units to Be Used in Systems with Limited Character Sets
X3.51-1986 Representations of Universal Time, Local Time Differentials, and United States Time Zone References
X3.52-1976 Unrecorded Single-Disk Cartridge (Front Loading, 2200 BPI) (General, Physical, and Magnetic Requirements)
X3.53-1976 Programming Language PL/I
X3.54-1986 Recorded Magnetic Tape (6250 CPI, Group Coded Recording)
X3.55-1982 Unrecorded Magnetic Tape Cartridge, 0.250 Inch (6.30 mm), 1600 bpi (63 bpmm), Phase encoded
X3.56-1986 Recorded Magnetic Tape Cartridge, 4 Track, 0.250 Inch (6.30 mm), 1600 bpi (63 bpmm), Phase Encoded
X3.57-1977 Structure for Formatting Message Headings Using the American National Standard Code for Information Interchange for Data Communication Systems Control
X3.58-1977 Unrecorded Eleven-Disk Pack (General, Physical, and Magnetic Requirements)
X3.60-1978 Programming Language Minimal BASIC
X3.61-1986 Representation of Geographic Point Locations
X3.62-1987 Paper Used in Optical Character Recognition (OCR) Systems
X3.63-1981 Unrecorded Twelve-Disk Pack (100 Megabytes) (General, Physical, and Magnetic Requirements)
X3.64-1979 Additional Controls for Use with American National Standard Code for Information Interchange
X3.66-1979 Advanced Data Communication Control Procedures (ADCCP)
X3.72-1981 Parallel Recorded Magnetic Tape Cartridge, 4 Track, 0.250 Inch (6.30 mm), 1600 bpi (63 bpmm), Phase Encoded
X3.73-1980 Single-Sided Unformatted Flexible Disk Cartridge (for 6631-BPR Use)
X3.74-1981 Programming Language PL/I, General-Purpose Subset
X3.76-1981 Unformatted Single-Disk Cartridge (Top Loading, 200 tpi 4400 bpi) (General, Physical, and Magnetic Requirements)
X3.77-1980 Representation of Pocket Select Characters
X3.78-1981 Representation of Vertical Carriage Positioning Characters in Information Interchange
X3.79-1981 Determination of Performance of Data Communications Systems That Use Bit-Oriented Communication Procedures
X3.80-1981 Interfaces between Flexible Disk Cartridge Drives and Their Host Controllers
X3.82-1980 One-Sided Single-Density Unformatted 5.25-Inch Flexible Disk Cartridge (for 3979-BPR Use)
X3.83-1980 ANSI Sponsorship Procedures for ISO Registration According to ISO 2375
X3.84-1981 Unformatted Twelve-Disk Pack (200 Megabytes)(General, Physical, and Magnetic Requirements
X3.85-1981 1/2-Inch Magnetic Tape Interchange Using a Self Loading Cartridge
X3.86-1980 Optical Character Recognition (OCR) Inks
X3.88-1981 Computer Program Abstracts
X3.89-1981 Unrecorded Single-Disk, Double-Density Cartridge (Front Loading, 2200 bpi, 200 tpi) (General, Physical, and Magnetic Requirements)
X3.91M-1987 Storage Module Interfaces
X3.92-1981 Data Encryption Algorithm
X3.93M-1981 OCR Character Positioning
X3.94-1985 Programming Language PANCM
X3.95-1982 Microprocessors — Hexadecimal Input/Output, Using 5-Bit and 7-Bit Teleprinters
X3.96-1983 Continuous Business Forms (Single-Part)
X3.98-1983 Text Information Interchange in Page Image Format (PIF)
X3.99-1983 Print Quality Guideline for Optical Character Recognition (OCR)
X3.100-1983 Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment for Packet Mode Operation with Packet Switched Data Communications Network
X3.101-1984 Interfaces Between Rigid Disk Drive(s) and Host(s)
X3.102-1983 Data Communication Systems and Services — User-Oriented Performance Parameters
X3.103-1983 Unrecorded Magnetic Tape Minicassette for Information Interchange, Coplanar 3.81 mm (0.150 in)
X3.104-1983 Recorded Magnetic Tape Minicassette for Information Interchange, Coplanar 3.81 mm (0.150 in), Phase Encoded
X3.105-1983 Data Link Encryption
X3.106-1983 Modes of Operation for the Data Encryption Algorithm
X3.110-1983 Videotex/Teletext Presentation Level Protocol Syntax
X3.111-1986 Optical Character Recognition (OCR) Matrix Character Sets for OCR-M
X3.112-1984 14-in (356-mm) Diameter Low-Surface-Friction Magnetic Storage Disk
X3.113-1987 Programming Language FULL BASIC
X3.114-1984 Alphanumeric Machines; Coded Character Sets for Keyboard Arrangements in ANSI X4.23-1982 and X4.22-1983

(Continued on reverse)