



BS 6964 : 1990
ISO/IEC 9075 : 1989

© British Standards Institution. No part of this publication may be photocopied or otherwise reproduced without the prior permission in writing of BSI

British Standard Specification for

Database language SQL with integrity enhancement

Langage de base de données SQL contenant l'amélioration sur l'intégrité — Spécifications
Datenbanksprache SQL mit verbesserter Datenintegrität

British Standards Institution

 **Documents**
Document Sharing Hub

National foreword

This British Standard has been prepared under the direction of the Information Systems Technology Standards Policy Committee. It is identical with ISO/IEC 9075 : 1989 'Information processing systems — Database language SQL with integrity enhancement', published by the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). It supersedes BS 6964 : 1988, which is withdrawn.

The international standard is a revision of ISO 9075 : 1987 which incorporates the text of draft addendum 1 on integrity enhancement.

The UK actively participated through BSI Technical Committee IST/21 in the development of the international standard.

Up-to-date information on existing and projected British Standards publications for Open Systems Interconnection is included in the 'Basic List', published as part of the BSI Information Technology Services (BITS) initiative. For details contact: Marketing Department, BSI, Linford Wood, Milton Keynes MK14 6LE.

Cross-reference

International standard	Corresponding British Standard
ISO 7185 : 1983	BS 6192 : 1982 Specification for computer programming language Pascal (Identical)

The Technical Committee has reviewed the provisions of ISO 1539, ISO 1989 and ISO 6160, to which reference is made in the text, and has decided that they are acceptable for use in conjunction with this standard.

Compliance with a British Standard does not of itself confer immunity from legal obligations.

Contents

National foreword	Inside front cover
Committees responsible	Back cover
1. Scope and field of application	1
2. References	3
3. Overview	5
3.1 Organization	5
3.2 Notation	5
3.3 Conventions	6
3.4 Conformance	6
4. Concepts	9
4.1 Sets	9
4.2 Data types	9
4.2.1 Character strings	9
4.2.2 Numbers	9
4.3 Columns	10
4.4 Tables	10
4.5 Integrity constraints	11
4.6 Schemas	11
4.7 The database	11
4.8 Modules	12
4.9 Procedures	12
4.10 Parameters	12
4.10.1 SQLCODE parameter	12
4.10.2 Indicator parameters	12
4.11 Standard programming languages	12
4.12 Cursors	13
4.13 Statements	14
4.14 Embedded syntax	14
4.15 Privileges	14
4.16 Transactions	15
5. Common elements	17
5.1 <character>	17
5.2 <literal>	18
5.3 <token>	20
5.4 Names	22
5.5 <data type>	24
5.6 <value specification> and <target specification>	26
5.7 <column specification>	28
5.8 <set function specification>	29
5.9 <value expression>	31
5.10 <predicate>	33
5.11 <comparison predicate>	34
5.12 <between predicate>	35
5.13 <in predicate>	36
5.14 <like predicate>	37
5.15 <null predicate>	39

5.16	< quantified predicate >	40
5.17	< exists predicate >	41
5.18	< search condition >	42
5.19	< table expression >	44
5.20	< from clause >	45
5.21	< where clause >	47
5.22	< group by clause >	48
5.23	< having clause >	49
5.24	< subquery >	50
5.25	< query specification >	52
6.	Schema definition language	55
6.1	< schema >	55
6.2	< table definition >	56
6.3	< column definition >	57
6.4	< default clause >	59
6.5	< table constraint definition >	61
6.6	< unique constraint definition >	62
6.7	< referential constraint definition >	63
6.8	< check constraint definition >	65
6.9	< view definition >	66
6.10	< privilege definition >	68
7.	Module language	71
7.1	< module >	71
7.2	< module name clause >	72
7.3	< procedure >	73
8.	Data manipulation language	77
8.1	< close statement >	77
8.2	< commit statement >	78
8.3	< declare cursor >	79
8.4	< delete statement: positioned >	83
8.5	< delete statement: searched >	84
8.6	< fetch statement >	85
8.7	< insert statement >	87
8.8	< open statement >	90
8.9	< rollback statement >	91
8.10	< select statement >	92
8.11	< update statement: positioned >	94
8.12	< update statement: searched >	96
9.	Levels	99
Annexes		103
Annex A.	< embedded SQL host program >	103
Annex B.	< embedded exception declaration >	107
Annex C.	< embedded SQL COBOL program >	109
Annex D.	< embedded SQL FORTRAN program >	111

Annex E. <embedded SQL Pascal program>	113
Annex F. <embedded SQL PL/I program>	115
Index	117

Information processing systems — Database Language SQL with integrity enhancement

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 and a separate integrity enhancement feature. Level 2 is the complete SQL database language excluding the integrity enhancement feature. Level 1 is the subset of Level 2 defined in clause 9, "Levels" on page 99.

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

The integrity enhancement feature comprises a means of specifying:

- 1) referential constraints between tables, which have to be satisfied;
- 2) check constraints to be applied to the rows of a table; and,
- 3) a default value for a column when a row is inserted into a table.

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

ISO/IEC 9075:1989(F)

library systems, and distributed communication systems, as well as various tools for database design, data administration, and performance optimization.

2. References

ISO 1539, *Programming Languages - FORTRAN*, (endorsement by ISO of American National Standard X3.9).

ISO 1989, *Programming Languages - COBOL*, (endorsement by ISO of American National Standard X3.23).

ISO 6160, *Programming Languages - PL/I*, (endorsement by ISO of American National Standard X3.53).

ISO 7185, *Programming Languages - Pascal*, (endorsement by ISO of British Standard 6192 for the English text).

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 6 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 17 defines language elements that occur in several parts of SQL language.
- 5) Clause 6, "Schema definition language" on page 55 defines the SQL facilities for specifying a database.
- 6) Clause 7, "Module language" on page 71 defines SQL modules and procedures.
- 7) Clause 8, "Data manipulation language" on page 77 defines the data manipulation statements of SQL.
- 8) Clause 9, "Levels" on page 99 defines the two levels of SQL and the integrity enhancement feature.

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 $\langle A \rangle$ is defined to "contain" a production symbol $\langle B \rangle$ if $\langle B \rangle$ occurs someplace in the expansion of $\langle A \rangle$. If $\langle A \rangle$ contains $\langle B \rangle$, then $\langle B \rangle$ is "contained in" $\langle A \rangle$. If $\langle A \rangle$ contains $\langle B \rangle$, then $\langle A \rangle$ is the "containing" $\langle A \rangle$ production symbol for $\langle B \rangle$.

ISO/IEC 9075:1989(E)

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 17 through clause 8, "Data manipulation language" on page 77 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>);

6 Database Language SQL with integrity enhancement

- 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 >).
- 3) Whether the integrity enhancement feature is implemented.

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

Previous page
is blank

ISO/IEC 9075:1989(F)

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 34.

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 $-N$.

ISO/IEC 9075:1989(E)

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.

Integrity constraints are effectively checked after execution of each <SQL statement>. If a 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.

A <unique constraint definition> requires that no two rows in a table have the same values in a specified column or columns.

A NOT NULL specification requires that none of the values in a column be the null value.

A <referential constraint definition> requires that for each row of one specified table, the "referencing table", the values of a specified column or columns either have at least one null value or be the same as the values of a specified column or columns in some row of another specified table, the "referenced table". The referencing table may be the same table as the referenced table.

A <check constraint definition> requires that a specified <search condition> not be false of any row of a table.

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.

ISO/IEC 9075:1989(E)

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 15) 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 include an `<order by clause>`, or if it includes an `<order by clause>` that does not completely specify the order of the rows, then the rows of the table have an order that is wholly or partly implementor-defined.

Within one transaction, where the order of rows is not defined or not wholly defined by an `<order by clause>`, the relative position of two rows will generally be the same each time the cursor is opened. The order may differ from one time to another if database values or parameter values referenced in the `<query specification>` in the `<declare cursor>` are different.

In separate transactions, even when the database values and parameter values referenced in the `<query specification>` of the `<declare cursor>` are the same, the order of rows designated by the same `<cursor specification>` and `<open statement>` may be different.

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, UPDATE, and REFERENCES.

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:

- 1) The effects of changes to the database and its contents made explicitly by the transaction itself.
- 2) The effects of differences in parameter values supplied to procedures that open cursors on subsequent retrievals through such cursors.

Previous page
is blank

ISO/IEC 9075:1989(E)

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 20.

- 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.

ISO/IEC 9075:1989(E)

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
    | CHARACTER | CHECK | CLOSE | COBOL | COMMIT
    | CONTINUE | COUNT | CREATE | CURRENT | CURSOR
    | DEC | DECIMAL | DECLARE | DEFAULT | DELETE | DESC | DISTINCT | DOUBLE
    | END | ESCAPE | EXEC | EXISTS
    | FETCH | FLOAT | FOR | FOREIGN | FORTRAN | FOUND | FROM
    | GO | GOTO | GRANT | GROUP | HAVING
    | IN | INDICATOR | INSERT | INT | INTEGER | INTO | IS
    | KEY | LANGUAGE | LIKE
    | MAX | MIN | MODULE | NOT | NULL | NUMERIC
    | OF | ON | OPEN | OPTION | OR | ORDER
    | PASCAL | PLI | PRECISION | PRIMARY | PRIVILEGES | PROCEDURE | PUBLIC
    | REAL | REFERENCES | 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> ::=
 <comment introducer> [<character> ...] <newline>

<comment introducer> ::= --See Syntax Rule 5.

<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>.
- 5) A <comment introducer> is a sequence of two or more consecutive hyphens (-) not separated by at least one <space> or <newline> and not contained in a <character string literal>.

General Rules

None.

ISO/IEC 9075:1989(E)

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 specification> (see 5.20, "<from clause>" on page 45). 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.

ISO/IEC 9075:1989(E)

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 <embedded 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 <correlation name> is implicit.
- NOTE:** The "scope" of a <table name> or <correlation name> is specified in 5.20, "<from clause>" on page 45, 6.2, "<table definition>" on page 56, 8.5, "<delete statement: searched>" on page 84, 8.11, "<update statement: positioned>" on page 94, and 8.12, "<update statement: searched>" on page 96.
- 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 44, 5.24, "<subquery>" on page 50, and 5.25, "<query specification>" on page 52.
- 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 28.

- 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 28.

- 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 34.
 - 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 $\langle \text{primary} \rangle$ is the null value, then the result of the $\langle \text{value expression} \rangle$ is the null value.
- 2) If operators are not specified, then the result of the $\langle \text{value expression} \rangle$ is the value of the specified $\langle \text{primary} \rangle$.
- 3) When a $\langle \text{value expression} \rangle$ 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.

ISO/IEC 9075:1989(E)

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 cardinality of the result of the < subquery > shall not be greater than one.
- 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.