INTERNATIONAL STANDARD

ISO/IEC 9075-1 9075-2 9075-5

First edition
1999-12-01

AMENDMENT 1
2001-03-15

Information technology — Database languages — SQL —

Part 1:
Framework (SQL/Framework)
Part 2:
Foundation (SQL/Foundation)
Part 5:
Host Language Bindings (SQL/Bindings)

AMENDMENT 1: On-Line Analytical Processing (SQL/OLAP)

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

Partie 1: Charpente (SQL/Charpente)

Partie 2: Fondations (SQL/Fondations)

Partie 5: Liants de langage d'hôte (SQL/Liants)

AMENDEMENT 1: Traitement analytique en ligne (SQL/OLAP)

Reference number
ISO/IEC 9075 (parts 1, 2 and 5):1999/Amd1:2001(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.ch
Web www.iso.ch

Printed in Switzerland

Contents

FIGURES

TABLES

Tables **Page**

Foreword

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

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 3.

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

Attention is drawn to the possibility that some of the elements of this Amendment may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to parts 1, 2 and 5 of ISO/IEC 9075:1999 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

Introduction

The organization of this Amendment is as follows:

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

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

3) Clause 3, "Definitions, notations, and conventions", defines the notations and conventions used in this Amendment.

4) Clause 4, "Concepts", presents concepts used in the definition of On-Line Analytical Processing facilities.

5) Clause 5, "Lexical elements", defines a number of lexical elements used in the definition of On-Line Analytical Processing facilities.

6) Clause 6, "Scalar expressions", defines a number of scalar expressions used in the definition of On-Line Analytical Processing facilities.

7) Clause 7, "Query expressions", defines the elements of the language that produce rows and tables of data as used in On-Line Analytical Processing facilities.

8) Clause 8, "Additional common elements", defines additional common elements used in the definition of On-Line Analytical Processing facilities.

9) Clause 9, "Schema definition and manipulation", defines the schema definition and manipulation statements associated with the definition of On-Line Analytical Processing facilities.

10) Clause 10, "SQL-client modules", defines SQL-client modules and externally-invoked procedures.

11) Clause 11, "Data manipulation", defines data manipulation operations associated with On-Line Analytical Processing facilities.

12) Clause 12, "Dynamic SQL", defines the SQL dynamic statements.

13) Clause 13, "Information Schema", defines viewed tables that contain schema information related to On-Line Analytical Processing facilities.

14) Clause 14, "Status codes", defines SQLSTATE values related to On-Line Analytical Processing facilities.

15) Clause 15, "Conformance", defines the criteria for conformance to this Amendment.

16) Annex A, "SQL conformance summary", is an informative Annex. It summarizes the conformance requirements of the SQL language.

17) Annex B, "Implementation-defined elements", is an informative Annex. It lists those features for which the body of this Amendment states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-defined.

18) Annex C, "Implementation-dependent elements", is an informative Annex. It lists those features for which the body of this Amendment states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or any other behavior is partly or wholly implementation-dependent.

19) Annex D, "SQL feature and package taxonomy", is an informative Annex. It identifies features of the SQL language specified in this Amendment by a numeric identifier and a short descriptive name. This taxonomy is used to specify conformance to Core SQL and may be used to develop other profiles involving the SQL language.

20) Annex E, "SQL Packages", is an informative Annex. It specifies a package of SQL language features.

In the text of this Amendment, Clauses begin a new odd-numbered page, and in Clause 5, "Lexical elements", through Clause 15, "Conformance", Subclauses begin a new page. Any resulting blank space is not significant.

Information technology — Database languages — SQL —

Part 1: Framework (SQL/Framework)
Part 2: Foundation (SQL/Foundation)
Part 5: Host Language Bindings (SQL/Bindings)

AMENDMENT 1:
On-Line Analytical Processing (SQL/OLAP)

1 Scope

This Amendment specifies the syntax and semantics of database language facilities that support on-line analytical processing.

The database language facilities that support on-line analytical processing include:

— Rank functions.

— Distribution functions.

— Inverse distribution functions (percentiles).

— Hypothetical set functions.

— Cumulative and other forms of moving aggregates.

— Variance, standard deviation, covariance, correlation, and linear regression functions.

This amendment also incidentally defines several new numeric functions.

NOTE 1 – The context for this Amendment is described by the Reference Model of Data Management (ISO/IEC 10032:1993).

(Blank page)

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this Amendment. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this Amendment are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards.

ISO/IEC 9075-1:1999, *Information technology — Database languages — SQL — Part 1: Framework (SQL/Framework).*

ISO/IEC 9075-2:1999, *Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation).*

ISO/IEC 9075-5:1999, *Information technology — Database languages — SQL — Part 5: Host Language Bindings (SQL/Bindings).*

(Blank page)

3 Definitions, notations, and conventions

3.1 Definitions

Insert this paragraph For the purposes of this Amendment, the definitions given in ISO/IEC 9075-1 and ISO/IEC 9075-2 apply.

3.2 Notation

Insert this paragraph The syntax notation used in this Amendment is an extended version of BNF ("Backus Normal Form" or "Backus Naur Form"). This version of BNF is fully described in Subclause 3.2, "Notation", of ISO/IEC 9075-1.

3.3 Conventions

Insert this paragraph Except as otherwise specified in this Amendment the conventions used in this Amendment are identical to those described in ISO/IEC 9075-1 and ISO/IEC 9075-2.

3.3.1 Use of terms

3.3.1.1 Syntactic containment

Replace 2nd paragraph *A1 directly contains B1* if *A1* contains *B1* without an intervening <subquery>, <within group specification>, or <set function specification> that is not an <ordered set function>.

3.3.2 Relationships to other parts of ISO/IEC 9075

3.3.2.1 Clause, Subclause, and Table relationships

Table 1—Clause, Subclause, and Table relationships

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Clause 1, “Scope”	Clause 1, "Scope"	ISO/IEC 9075-2
Clause 2, “Normative references”	Clause 2, "Normative references"	ISO/IEC 9075-2
Clause 3, “Definitions, notations, and conventions”	Clause 3, "Definitions, notations, and conventions"	ISO/IEC 9075-2
Subclause 3.1, “Definitions”	Subclause 3.1, "Definitions"	ISO/IEC 9075-2
Subclause 3.2, “Notation”	Subclause 3.2, "Notation"	ISO/IEC 9075-2

3.3 Conventions

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 3.3, “Conventions”	Subclause 3.3, "Conventions"	ISO/IEC 9075-2
Subclause 3.3.1, “Use of terms”	Subclause 3.3.1, "Use of terms"	ISO/IEC 9075-2
Subclause 3.3.1.1, “Syntactic containment”	Subclause 3.3.1.1, "Syntactic containment"	ISO/IEC 9075-2
Subclause 3.3.2, “Relationships to other parts of ISO/IEC 9075”	*(none)*	*(none)*
Subclause 3.3.2.1, “Clause, Subclause, and Table relationships”	*(none)*	*(none)*
Clause 4, “Concepts”	Clause 4, "Concepts"	ISO/IEC 9075-2
Subclause 4.1, “Numbers”	Subclause 4.5, "Numbers"	ISO/IEC 9075-2
Subclause 4.1.1, “Operations involving numbers”	Subclause 4.5.2, "Operations involving numbers"	ISO/IEC 9075-2
Subclause 4.2, “Tables”	Subclause 4.16, "Tables"	ISO/IEC 9075-2
Subclause 4.2.1, “Windowed tables”	*(none)*	*(none)*
Subclause 4.3, “Data analysis operations (involving tables)”	*(none)*	*(none)*
Subclause 4.3.1, “Group functions”	*(none)*	*(none)*
Subclause 4.3.2, “Window functions”	*(none)*	*(none)*
Subclause 4.3.3, “Aggregate functions”	*(none)*	*(none)*
Clause 5, “Lexical elements”	Clause 5, "Lexical elements"	ISO/IEC 9075-2
Subclause 5.1, “<token> and <separator>”	Subclause 5.2, "<token> and <separator>"	ISO/IEC 9075-2
Subclause 5.2, “Names and identifiers”	Subclause 5.4, "Names and identifiers"	ISO/IEC 9075-2
Clause 6, “Scalar expressions”	Clause 6, "Scalar expressions"	ISO/IEC 9075-2
Subclause 6.1, “<set function specification>”	Subclause 6.16, "<set function specification>"	ISO/IEC 9075-2
Subclause 6.2, “<numeric value function>”	Subclause 6.17, "<numeric value function>"	ISO/IEC 9075-2
Subclause 6.3, “<window function>”	*(none)*	*(none)*
Subclause 6.4, “<value expression>”	Subclause 6.23, "<value expression>"	ISO/IEC 9075-2
Clause 7, “Query expressions”	Clause 7, "Query expressions"	ISO/IEC 9075-2
Subclause 7.1, “<table expression>”	Subclause 7.4, "<table expression>"	ISO/IEC 9075-2
Subclause 7.2, “<joined table>”	Subclause 7.7, "<joined table>"	ISO/IEC 9075-2

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Subclause 7.3, “<where clause>”	Subclause 7.8, "<where clause>"	ISO/IEC 9075-2
Subclause 7.4, “<having clause>”	Subclause 7.10, "<having clause>"	ISO/IEC 9075-2
Subclause 7.5, “<window clause>”	*(none)*	*(none)*
Subclause 7.6, “<query specification>”	Subclause 7.11, "<query specification>"	ISO/IEC 9075-2
Clause 8, “Additional common elements”	Clause 10, "Additional common elements"	ISO/IEC 9075-2
Subclause 8.1, “<aggregate function>”	*(none)*	*(none)*
Subclause 8.2, “<sort specification list>”	*(none)*	*(none)*
Clause 9, “Schema definition and manipulation”	Clause 11, "Schema definition and manipulation"	ISO/IEC 9075-2
Subclause 9.1, “<drop table constraint definition>”	Subclause 11.19, "<drop table constraint definition>"	ISO/IEC 9075-2
Subclause 9.2, “<drop user-defined ordering statement>”	Subclause 11.55, "<drop user-defined ordering statement>"	ISO/IEC 9075-2
Clause 10, “SQL-client modules”	Clause 13, "SQL-client modules"	ISO/IEC 9075-2
Subclause 10.1, “Calls to an <externally-invoked procedure>”	Subclause 13.4, "Calls to an <externally-invoked procedure>"	ISO/IEC 9075-2
Clause 11, “Data manipulation”	Clause 14, "Data manipulation"	ISO/IEC 9075-2
Subclause 11.1, “<declare cursor>”	Subclause 14.1, "<declare cursor>"	ISO/IEC 9075-2
Subclause 11.2, “<select statement: single row>”	Subclause 14.5, "<select statement: single row>"	ISO/IEC 9075-2
Clause 12, “Dynamic SQL”	Clause 15, "Dynamic SQL"	ISO/IEC 9075-5
Subclause 12.1, “<prepare statement>”	Subclause 15.6, "<prepare statement>"	ISO/IEC 9075-5
Clause 13, “Information Schema”	Clause 20, "Information Schema"	ISO/IEC 9075-2
Subclause 13.1, “Definition of SQL built-in functions”	Subclause 20.70, "Definition of SQL built-in functions"	ISO/IEC 9075-2
Clause 14, “Status codes”	Clause 22, "Status codes"	ISO/IEC 9075-2
Subclause 14.1, “SQLSTATE”	Subclause 22.1, "SQLSTATE"	ISO/IEC 9075-2
Clause 15, “Conformance”	Clause 23, "Conformance"	ISO/IEC 9075-2
Subclause 15.1, “General conformance requirements”	Subclause 23.1, "General conformance requirements"	ISO/IEC 9075-2
Annex A, “SQL conformance summary”	Annex A, "SQL Conformance Summary"	ISO/IEC 9075-2

3.3 Conventions

Table 1—Clause, Subclause, and Table relationships (Cont.)

Clause, Subclause, or Table in this part of ISO/IEC 9075	Corresponding Clause, Subclause, or Table from another part	Part containing correspondence
Annex B, “Implementation-defined elements”	Annex B, "Implementation-defined elements"	ISO/IEC 9075-2
Annex C, “Implementation-dependent elements”	Annex C, "Implementation-dependent elements"	ISO/IEC 9075-2
Annex D, “SQL feature and package taxonomy”	Annex F, "SQL feature and package taxonomy"	ISO/IEC 9075-2
Annex E, “SQL Packages”	Annex B, "SQL Packages"	ISO/IEC 9075-1
Subclause E.1, “OLAP”	*(none)*	*(none)*
Figure 1, “Illustration of WIDTH_BUCKET Semantics”	*(none)*	*(none)*
Table 1, “Clause, Subclause, and Table relationships”	*(none)*	*(none)*
Table 2, “SQLSTATE class and subclass values”	Table 27, "SQLSTATE class and subclass values"	ISO/IEC 9075-2
Table 3, “Implied feature relationships”	Table 30, "Implied feature relationships"	ISO/IEC 9075-2
Table 4, “SQL/OLAP feature taxonomy for features outside Core SQL”	Table 32, "SQL/Foundation feature taxonomy for features outside Core SQL"	ISO/IEC 9075-2
Table 5, “SQL Packages”	Table 2, "SQL Packages"	ISO/IEC 9075-1

4 Concepts

4.1 Numbers

4.1.1 Operations involving numbers

Insert this paragraph The following are also functions that return numbers:

— <natural logarithm> computes the natural logarithm of its argument.

— <exponential function> computes the exponential function, that is, *e*, (the base of natural logarithms) raised to the power equal to its argument.

— <power function> raises its first argument to the power of its second argument.

— <square root> computes the square root of its argument.

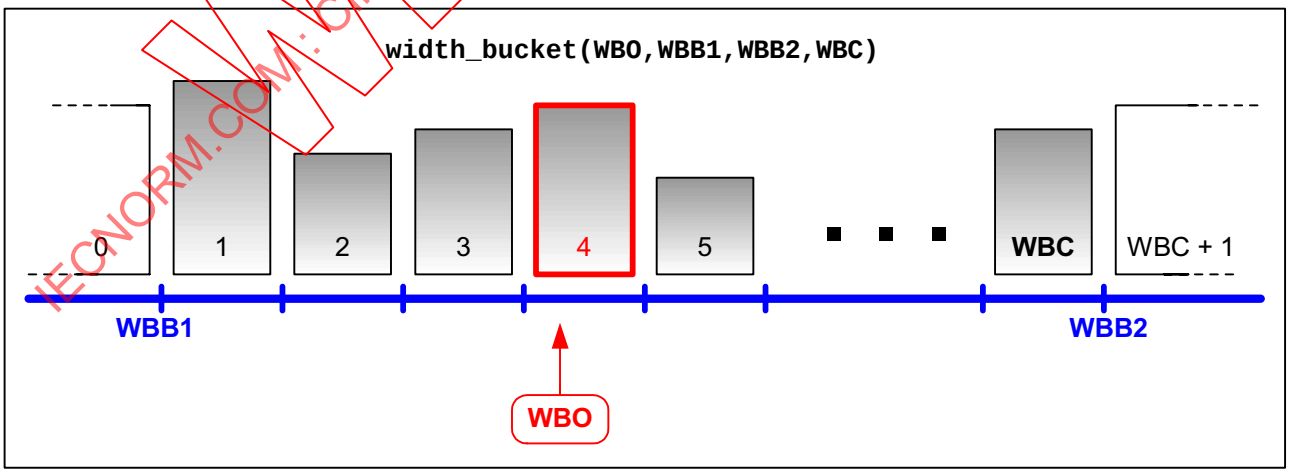
— <floor function> computes the greatest integer less than or equal to its argument.

— <ceiling function> computes the least integer greater than or equal to its argument.

— <width bucket> is a function of four arguments, returning an integer between 0 (zero) and the value of the final argument plus 1 (one), by assigning the first argument to an equi-width partitioning of the range of numbers between the second and third arguments. Values outside the range between the second and third arguments are assigned to either 0 (zero) or the value of the final argument plus 1 (one).

NOTE 2 – The semantics of <width bucket> are illustrated in Figure 1, "Illustration of WIDTH_BUCKET Semantics".

Figure 1—Illustration of WIDTH_BUCKET Semantics



4.2 Tables

4.2.1 Windowed tables

A *windowed table* is a table together with one or more windows. A *window* is a transient data structure associated with a <table expression>. A window is defined explicitly by a <window definition> or implicitly by an <inline window specification>. Implicitly defined windows have an implementation-dependent window name. A window is used to specify window partitions and window frames, which are multisets of rows used in the definition of <window function>s.

Every window defines a *window partitioning* of the rows of the <table expression>. The window partitioning is specified by a list of columns. Window partitioning is similar to forming groups of a grouped table. However, unlike grouped tables, each row is retained in the result of the <table expression>. The *window partition* of a row *R* is the multiset of rows *R2* that are not distinct from *R*, for all columns enumerated in the window partitioning clause. The window partitioning clause is optional; if omitted, there is a single window partition consisting of all the rows in the result.

If a <table expression> is grouped and also has a window, then there is a syntactic transformation that segregates the grouping into a <derived table>, so that the window partitions consist of rows of the <derived table> rather than groups of rows.

A window may define a *window ordering* of rows within each window partition defined by the window. The window ordering of rows within window partitions is specified by a list of <value expression>s, followed by ASC (for ascending order) or DESC (for descending order). In addition, NULLS FIRST or NULLS LAST may be specified, to indicate whether a null value should appear before or after all non-null values in the ordered sequence of each <value expression>.

Optionally, a window may define a *window frame* for each row *R*. A window frame is always defined relative to the current row. A window frame is specified by up to four syntactic elements:

— The choice of RANGE, to indicate a logical definition of the window frame by offsetting forward or backward from the current row by an increment or decrement to the sort key; or ROWS, to indicate a physical definition of the window frame, by counting rows forward or backward from the current row.

— A starting row, which may be the first row of the window partition of *R*, the current row, or some row determined by a logical or physical offset from the current row.

— An ending row, which may be the last row of the window partition of *R*, the current row, or some row determined by a logical or physical offset from the current row.

— A <window frame exclusion>, indicating whether to exclude the current row and/or its peers (if not already excluded by being prior to the starting row or after the ending row).

A window is described by a *window structure descriptor*, including:

— The window name.

— Optionally, the ordering window name—that is, the name of another window, called the *ordering window*, that is used to define the partitioning and ordering of the present window.

— The window partitioning clause—that is, a <window partition clause>, if any is specified in either the present <window specification> or in the window descriptor of the ordering window.

— The window ordering clause—that is, a <window order clause>, if any is specified in either the present <window specification> or in the window descriptor of the ordering window.

— The window framing clause—that is, a <window frame clause>, if any.

In general, two <window function>s are computed independently, each one performing its own sort of its data, even if they use the same data and the same <sort specification list>. Since sorts may specify partial orderings, the computation of <window function>s is inevitably non-deterministic to the extent that the ordering is not total. Nevertheless, the user may desire that two <window function>s be computed using the same ordering, so that, for example, two moving aggregates move through the rows of a partition in precisely the same order. Two <window function>s are computed using the same (possibly non-deterministic) window ordering of the rows if any of the following are true:

— The <window function>s identify the same window structure descriptor.

— The <window function>s' window structure descriptors have window partitioning clauses that enumerate the same number of column references, and those column references are pairwise equivalent in their order of occurrence; and their window structure descriptors have window ordering clauses with the same number of <sort key>s, and those <sort key>s are all column references, and those column references are pairwise equivalent in their order of occurrence, and the <sort specification>s pairwise specify or imply <collate clause>s that specify equivalent <collation name>s, the same <ordering specification> (ASC or DESC), and the same <null ordering> (NULLS FIRST or NULLS LAST).

— The window structure descriptor of one <window function> is the ordering window of the other <window function>, or both window structure descriptors identify the same ordering window.

4.3 Data analysis operations (involving tables)

A data analysis function is a function that returns a value derived from a number of rows in the result of a <table expression>. A data analysis function may only be invoked as part of a <query specification> or <select statement: single row>, and then only in certain contexts, identified below. A data analysis function is one of:

— A group function, which is invoked on a grouped table and computes a grouping operation or an aggregate function from a group of the grouped table.

— A window function, which is invoked on a windowed table and computes a rank, row number or window aggregate function.

4.3.1 Group functions

A group function may only appear in the <select list>, <having clause> or <window clause> of a <query specification> or <select statement: single row>, or in the <order by clause> of a cursor that is a simple table query.

A group function is one of:

— The *grouping operation*.

— A *group aggregate function*.

4.3 Data analysis operations (involving tables)

The grouping operation is of the form GROUPING(<column reference>). The result of such an invocation is 1 (one) in the case of a row whose values are the results of aggregation over that <column reference> during the execution of a grouped query containing CUBE, ROLLUP, or GROUPING SET, and 0 (zero) otherwise.

4.3.2 Window functions

A window function is a function whose result for a given row is derived from the window frame of that row as defined by a window structure descriptor of a windowed table. Window functions may only appear in the <select list> of a <query specification> or <select statement: single row>, or the <order by clause> of a simply table query.

A window function is one of:

— A rank function.

— A distribution function.

— The row number function.

— A window aggregate function.

The rank functions compute the ordinal rank of a row *R* within the window partition of *R* as defined by a window structure descriptor, according to the window ordering of those rows, also specified by the same window structure descriptor. Rows that are not distinct with respect to the window ordering within their window partition are assigned the same rank. There are two variants, indicated by the keywords RANK and DENSE_RANK.

— If RANK is specified, then the rank of row *R* is defined as 1 (one) plus the number of rows that precede *R* and are not peers of *R*.

NOTE 3 – This implies that if two or more rows are not distinct with respect to the window ordering, then there will be one or more gaps in the sequential rank numbering.

— If DENSE_RANK is specified, then the rank of row *R* is defined as the number of rows preceding and including *R* that are distinct with respect to the window ordering.

NOTE 4 – This implies that there are no gaps in the sequential rank numbering of rows in each window partition.

The distribution functions compute a relative rank of a row *R* within the window partition of *R* defined by a window structure descriptor, expressed as an approximate numeric ratio between 0.0 and 1.0. There are two variants, indicated by the keywords PERCENT_RANK and CUME_DIST.

— If PERCENT_RANK is specified, then the relative rank of a row *R* is defined as $(RK-1)/(NR-1)$, where *RK* is defined to be the RANK of *R* and *NR* is defined to be the number of rows in the window partition of *R*.

— If CUME_DIST is specified, then the relative rank of a row *R* is defined as *NP*/*NR*, where *NP* is defined to be the number of rows preceding or peer with *R* in the window ordering of the window partition of *R* and *NR* is defined to be the number of rows in the window partition of *R*.

The ROW_NUMBER function computes the sequential row number, starting with 1 (one) for the first row, of the row within its window partition according to the window ordering of the window.

The window aggregate functions compute an aggregate value (COUNT, SUM, AVG, *etc.*), the same as a group aggregate function, except that the computation aggregates over the window frame of a row rather than over a group of a grouped table. The hypothetical set functions are not permitted as window aggregate functions.

4.3.3 Aggregate functions

An aggregate function is a function whose result is derived from an aggregation of rows defined by one of:

— The grouping of a grouped table, in which case the aggregate function is a group aggregate function, or set function, and for each group there is one aggregation, which includes every row in the group.

— The window frame of a row *R* of a windowed table relative to a particular window structure descriptor, in which case the aggregate function is a window aggregate function, and the aggregation consists of every row in the window frame of *R*, as defined by the window structure descriptor.

Optionally, the multiset of rows in an aggregation may be filtered, retaining only those rows that satisfy a <search condition> that is specified by a <filter clause>.

The result of the aggregate function COUNT (*) is the number of rows in the aggregation.

Every other aggregate function may be classified as a *unary aggregate function*, a *binary aggregate function*, an *inverse distribution*, or a *hypothetical set function*.

Every unary aggregate function takes an arbitrary <value expression> as the argument; most unary aggregate functions can optionally be qualified with either DISTINCT or ALL. Of the rows in the aggregation, the following do not qualify:

— If DISTINCT is specified, then redundant duplicates.

— Every row in which the <value expression> evaluates to the null value.

If no row qualifies, then the result of COUNT is 0 (zero), and the result of any other aggregate function is the null value.

Otherwise (*i.e.*, at least one row qualifies), the result of the aggregate function is:

— If COUNT <value expression> is specified, then the number of rows that qualify.

— If SUM is specified, then the sum of <value expression> evaluated for each row that qualifies.

— If AVG is specified, then the average of <value expression> evaluated for each row that qualifies.

— If MAX is specified, then the maximum value of <value expression> evaluated for each row that qualifies.

— If MIN is specified, then the minimum value of <value expression> evaluated for each row that qualifies.

— If EVERY is specified, then true if the <value expression> evaluates to *true* for every row that qualifies, otherwise, *false* .

— If ANY or SOME is specified, then *true* if the <value expression> evaluates to *true* for at least one row remaining in the group; otherwise, *false* .

4.3 Data analysis operations (involving tables)

— If VAR_POP is specified, then the population variance of <value expression> evaluated for each row remaining in the group, defined as the sum of squares of the difference of <value expression> from the mean of <value expression>, divided by the number of rows remaining.

— If VAR_SAMP is specified, then the sample variance of <value expression> evaluated for each row remaining in the group, defined as the sum of squares of the difference of <value expression> from the mean of <value expression>, divided by the number of rows remaining minus 1 (one).

— If STDDEV_POP is specified, then the population standard deviation of <value expression> evaluated for each row remaining in the group, defined as the square root of the population variance.

— If STDDEV_SAMP is specified, then the sample standard deviation of <value expression> evaluated for each row remaining in the group, defined as the square root of the sample variance.

Neither DISTINCT nor ALL are allowed to be specified for VAR_POP, VAR_SAMP, STDDEV_POP or STDDEV_SAMP; redundant duplicates are not removed when computing these functions.

The binary aggregate functions take a pair of arguments, the <dependent variable expression> and the <independent variable expression>, which are both <numeric value expression>s. Any row in which either argument evaluates to the null value is removed from the group. If there are no rows remaining in the group, then the result of REGR_COUNT is 0 (zero), and the other binary aggregate functions result in the null value. Otherwise, the computation concludes and the result is:

— If REGR_COUNT is specified, then the number of rows remaining in the group.

— If COVAR_POP is specified, then the population covariance, defined as the sum of products of the difference of <independent variable expression> from its mean times the difference of <dependent variable expression> from its mean, divided by the number of rows remaining.

— If COVAR_SAMP is specified, then the sample covariance, defined as the sum of products of the difference of <independent variable expression> from its mean times the difference of <dependent variable expression> from its mean, divided by the number of rows remaining minus 1 (one).

— If CORR is specified, then the correlation coefficient, defined as the ratio of the population covariance divided by the product of the population standard deviation of <independent variable expression> and the population standard deviation of <dependent variable expression>.

— If REGR_R2 is specified, then the square of the correlation coefficient.

— If REGR_SLOPE is specified, then the slope of the least-squares-fit linear equation determined by the (<independent variable expression>, <dependent variable expression>) pairs.

— If REGR_INTERCEPT is specified, then the y-intercept of the least-squares-fit linear equation determined by the (<independent variable expression>, <dependent variable expression>) pairs.

— If REGR_SXX is specified, then the sum of squares of <independent variable expression>.

— If REGR_SYY is specified, then the sum of squares of <dependent variable expression>.

— If REGR_SXY is specified, then the sum of products of <independent variable expression> times <dependent variable expression>.

— If REGR_AVGX is specified, then the average of <independent variable expression>.

— If REGR_AVGY is specified, then the average of <dependent variable expression>.

There are two inverse distribution functions, PERCENTILE_CONT and PERCENTILE_DISC. Both inverse distribution functions specify an argument and an ordering of a value expression. The value of the argument should be between 0 (zero) and 1 (one) inclusive. The value expression is evaluated for each row of the group, nulls are discarded, and the remaining rows are ordered. The computation concludes:

— If PERCENTILE_CONT is specified, by considering the pair of consecutive rows that are indicated by the argument, treated as a fraction of the total number of rows in the group, and interpolating the value of the value expression evaluated for these rows.

— If PERCENTILE_DISC is specified, by treating the group as a window partition of the CUME_DIST window function, using the specified ordering of the value expression as the window ordering, and returning the first value expression whose cumulative distribution value is greater than or equal to the argument.

The hypothetical set functions are related to the window functions RANK, DENSE_RANK, PERCENT_RANK and CUME_DIST, and use the same names, though with a different syntax. These functions take an argument *A* and an ordering of a value expression *VE*. *VE* is evaluated for all rows of the group. This multiset of values is augmented with *A*; the resulting collection is treated as a window partition of the corresponding window function whose window ordering is the ordering of the value expression. The result of the hypothetical set function is the value of the eponymous window function for the hypothetical "row" that contributes *A* to the collection.

(Blank page)

5 Lexical elements

5.1 <token> and <separator>

Function

Specify lexical units (tokens and separators) that participate in SQL language.

Format

```
<non-reserved word> ::=
        !! All alternatives from ISO/IEC 9075-2
      | !! All alternatives from ISO/IEC 9075-5

      | CEIL | CEILING | CORR | COVAR_POP | COVAR_SAMP | CUME_DIST

      | DENSE_RANK

      | EXCLUDE | EXP

      | FILTER | FLOOR | FOLLOWING

      | LN

      | NULLS

      | OTHERS | OVER

      | PARTITION | PERCENTILE_CONT | PERCENTILE_DISC | PERCENT_RANK | POWER | PRECEDING


      | RANGE | RANK | REGR_AVGX | REGR_AVGY | REGR_COUNT | REGR_INTERCEPT
      | REGR_R2 | REGR_SLOPE | REGR_SXX | REGR_SXY | REGR_SYY | ROW_NUMBER

      | SQRT | STDDEV_POP | STDDEV_SAMP

      | TIES

      | UNBOUNDED

      | VAR_POP | VAR_SAMP

      | WIDTH_BUCKET

<reserved word> ::=
        !! All alternatives from ISO/IEC 9075-2
      | !! All alternatives from ISO/IEC 9075-5

      | WINDOW
```

5.1 <token> and <separator>

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

5.2 Names and identifiers

Function

Specify names.

Format

```
<window name> ::= <identifier>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR A <window name> identifies a window.

Conformance Rules

1) Insert this CR Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify <window name>.

(Blank page)

6 Scalar expressions

6.1 <set function specification>

Function

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

Format

```
<set function specification> ::=
        <aggregate function>
      | <grouping operation>
```

Delete the definition of <general set function>

Delete the definition of <set function type>

Delete the definition of <computational operation>

Delete the definition of <set quantifier>

NOTE 5 – The deleted definitions are now placed in Subclause 8.1, "<aggregate function>".

Syntax Rules

1) Delete SR 1) through 3)

2) Insert this SR If <aggregate function> specifies a <filter clause>, then the <filter condition> shall not contain a <set function specification>.

3) Replace SR 4) If <aggregate function> specifies a <general set function>, then the <value expression> simply contained in the <general set function> shall not contain a <set function specification> or a <subquery>. If the <value expression> contains a column reference that is an outer reference, then that outer reference shall be the only column reference contained in the <value expression>.

4) Insert this SR If <aggregate function> specifies <binary set function>, then neither the <dependent variable expression> nor the <independent variable expression> simply contained in the <binary set function> shall contain a <set function specification> or a <subquery>. If the <dependent variable expression> or <independent variable expression> contains a column reference that is an outer reference, then that outer reference shall be the only column reference contained in the <dependent variable expression> or <independent variable expression>.

5) Insert this SR A <value expression> *VE* simply contained in a <set function specification> *SFE* is an *aggregated argument* of *SFE* if either *SFE* is not an <ordered set function> or *VE* is simply contained in a <within group specification>; otherwise, *VE* is a *non-aggregated argument* of *SFE*.

6.1 <set function specification>

6) Replace SR 5) If an aggregated argument of a <set function specification> contains a column reference that is an outer reference, then the <set function specification> shall be contained in either:

   a) A <select list>.

   b) A <window clause>.

   c) A <subquery> of a <having clause>, in which case the qualifying table of the <column reference> shall be the table referenced by a <table reference> that is directly contained in the <table expression> that directly contains the <having clause>.

7) Insert this SR If <aggregate function> is specified, then the declared type of the result is the declared type of the <aggregate function>. If the declared type is character string, then the collating sequence and coercibility characteristic are the collating sequence and coercibility characteristic of the <aggregate function>.

8) Delete SRs 6) through 14)

9) Delete SR 16)

Access Rules

No additional Access Rules.

General Rules

1) Delete GR 1) and GR 2)

2) Delete GRs 3)a) through 3)g)

3) Insert this GR If <aggregate function> is specified, then the result is the value of the <aggregate function>.

Conformance Rules

1) Delete CRs 1) through 6)

2) Delete CRs 8) and 9)

6.2 <numeric value function>

Function

Specify a function yielding a value of type numeric.

Format

```
<numeric value function> ::=
        !! All alternatives from ISO/IEC 9075-2
      | <natural logarithm>
      | <exponential function>
      | <power function>
      | <square root>
      | <floor function>
      | <ceiling function>
      | <width bucket function>

<natural logarithm> ::=
      LN <left paren> <numeric value expression> <right paren>

<exponential function> ::=
      EXP <left paren> <numeric value expression> <right paren>

<power function> ::=
      POWER <left paren> <numeric value expression base>
          <comma> <numeric value expression exponent> <right paren>

<numeric value expression base> ::=
      <numeric value expression>

<numeric value expression exponent> ::=
      <numeric value expression>

<square root> ::=
      SQRT <left paren> <numeric value expression> <right paren>

<floor function> ::=
      FLOOR <left paren> <numeric value expression> <right paren>

<ceiling function> ::=
      { CEIL | CEILING } <left paren> <numeric value expression> <right paren>

<width bucket function> ::=
      WIDTH_BUCKET <left paren> <width bucket operand> <comma>
          <width bucket bound 1> <comma> <width bucket bound 2>
          <comma> <width bucket count> <right paren>

<width bucket operand> ::=
      <numeric value expression>

<width bucket bound 1> ::=
      <numeric value expression>

<width bucket bound 2> ::=
      <numeric value expression>

<width bucket count> ::=
      <numeric value expression>
```

6.2 <numeric value function>

Syntax Rules

1) Insert this SR The declared type of the result of <natural logarithm> is approximate numeric with implementation-defined precision.

2) Insert this SR The declared type of the result of <exponential function> is approximate numeric with implementation-defined precision.

3) Insert this SR The declared type of the result of <power function> is approximate numeric with implementation-defined precision.

4) Insert this SR If <square root> is specified, then let *NVE* be the simply contained <numeric value expression>. The <square root> is equivalent to

```
POWER (NVE, 0.5)
```

5) Insert this SR The declared type of the result of <floor function> is exact numeric with implementation-defined precision, with the radix of the simply contained <numeric value expression>, and with scale 0 (zero).

6) Insert this SR The declared type of the result of <ceiling function> is exact numeric with implementation-defined precision, with the radix of the simply contained <numeric value expression>, and with scale 0 (zero).

7) Insert this SR If <width bucket function> is specified, then the declared type of <width bucket count> shall be exact numeric with scale 0 (zero). The declared type of the result of <width bucket function> is the declared type of <width bucket count>.

Access Rules

No additional Access Rules.

General Rules

1) Insert this GR If <natural logarithm> is specified, then let *V* be the value of the simply contained <numeric value expression>.

   a) If *V* is the null value, then the result is the null value.

   b) If *V* is 0 (zero) or negative, then an exception condition is raised: *data exception — invalid argument for natural logarithm*.

   c) Otherwise, the result is the natural logarithm of *V*.

2) Insert this GR If <exponential function> is specified, then let *V* be the value of the simply contained <numeric value expression>.

   a) If *V* is the null value, then the result is the null value.

   b) Otherwise, the result is *e* (the base of natural logarithms) raised to the power *V*. If the result is not representable in the the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range*.

3) Insert this GR If <power function> is specified, then let *NVEB* be the <numeric value expression base>, then let *VB* be the value of *NVEB*, let *NVEE* be the <numeric value expression exponent>, and let *VE* be the value of *NVEE*.

   a) If either *VB* or *VE* is the null value, then the result is the null value.

   b) If *VB* is 0 (zero) and *VE* is negative, then an exception condition is raised: *data exception — invalid argument for power function*.

   c) If *VB* is 0 (zero) and *VE* is 0 (zero), then the result is 1 (one).

   d) If *VB* is 0 (zero) and *VE* is positive, then the result is 0 (zero).

   e) If *VB* is negative and *VE* is not equal to an exact numeric value with scale 0 (zero), then an exception condition is raised: *data exception — invalid argument for power function*.

   f) If *VB* is negative and *VE* is equal to an exact numeric value with scale 0 (zero) that is an even number, then the result is the result of

      `EXP(`*NVEE*`*LN(-`*NVEB*`))`

   g) If *VB* is negative and *VE* is equal to an exact numeric value with scale 0 (zero) that is an odd number, then the result is the result of

      `-EXP(`*NVEE*`*LN(-`*NVEB*`))`

   h) Otherwise, the result is the result of

      `EXP(`*NVEE*`*LN(`*NVEB*`))`

4) Insert this GR If <floor function> is specified, then let *V* be the value of the simply contained <numeric value expression>.

   Case:

   a) If *V* is the null value, then the result is the null value.

   b) Otherwise, the result is the greatest exact numeric value with scale 0 (zero) that is less than or equal to *V*. If this result is not representable by the result data type, then an exception condition is raised: *data exception — numeric value out of range*.

5) Insert this GR If <ceiling function> is specified, then let *V* be the value of the simply contained <numeric value expression>.

   Case:

   a) If *V* is the null value, then the result is the null value.

   b) Otherwise, the result is the least exact numeric value with scale 0 (zero) that is greater than or equal to *V*. If this result is not representable by the result data type, then an exception condition is raised: *data exception — numeric value out of range*.

6) Insert this GR If <width bucket function> is specified, then let *WBO* be the value of <width bucket operand>, let *WBB1* be the value of <width bucket bound 1>, let *WBB2* be the value of <width bucket bound 2>, and let *WBC* be the value of <width bucket count>.

   Case:

   a) If any of *WBO*, *WBB1*, *WBB2*, or *WBC* is the null value, then the result is the null value.

6.2 <numeric value function>

b) If *WBC* is less than or equal to 0 (zero), then an exception condition is raised: *data exception — invalid argument for width bucket function*.

c) If *WBB1* equals *WBB2*, then an exception condition is raised: *data exception — invalid argument for width bucket function*.

d) If *WBB1* is less than *WBB2*, then

Case:

i) If *WBO* is less than *WBB1*, then the result is 0 (zero).

ii) If *WBO* is greater than or equal to *WBB2*, then the result is *WBC*+1. If the result is not representable in the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range*.

iii) Otherwise, the result is the greatest exact numeric value with scale 0 (zero) that is less than or equal to ((*WBC* * (*WBO* - *WBB1*) / (*WBB2* - *WBB1*)) + 1).

e) If *WBB1* is greater than *WBB2*, then

Case:

i) If *WBO* is greater than *WBB1*, then the result is 0 (zero).

ii) If *WBO* is less than or equal to *WBB2*, then the result is *WBC*+1. If the result is not representable in the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range*.

iii) Otherwise, the result is the least exact numeric value with scale 0 (zero) that is less than or equal to ((*WBC* * (*WBB1* - *WBO*) / (*WBB1* - *WBB2*)) + 1).

Conformance Rules

The following restrictions apply for Core SQL:

1) Insert this CR Without Feature T621, "Enhanced numeric functions", conforming SQL language shall not specify <natural logarithm>, <exponential function>, <power function>, <square root>, <floor function> or <ceiling function>.

2) Insert this CR Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify <width bucket function>.

6.3 <window function>

Function

Specify a window function.

Format

```
<window function> ::=
       <window function type> OVER <window name or specification>

<window function type> ::=
         <rank function type> <left paren> <right paren>
       | ROW_NUMBER <left paren> <right paren>
       | <aggregate function>

<rank function type> ::=
         RANK
       | DENSE_RANK
       | PERCENT_RANK
       | CUME_DIST

<window name or specification> ::=
         <window name>
       | <in-line window specification>

<in-line window specification> ::=
       <window specification>
```

Syntax Rules

1) An <aggregate function> simply contained in a <window function> shall not simply contain a <hypothetical set function>.

2) Let *OF* be the <window function>.

3) Case:

   a) If *OF* is contained in an <order by clause>, then the <order by clause> shall be contained in a <cursor specification> that is a simple table query. Let *ST* be the sort table that is obtained by applying the syntactic transformation of a simple table query, as specified in Subclause 11.1, "<declare cursor>". Let *TE* be the <table expression> contained in the result of that syntactic transformation.

   b) Otherwise, *OF* shall be contained in a <select list> that is immediately contained in a <query specification> *QS* or a <select statement: single row> *SSSR*. Let *QSS* be the innermost <query specification> contained in *QS* that contains *OF*. Let *TE* be the <table expression> immediately contained in *QSS* or *SSSR*.

4) *OF* shall not contain an outer reference or a <subquery>.

5) If the window ordering clause or the window framing clause of the window structure descriptor that describes the <window name or specification> is present, then no <aggregate function> simply contained in <window function> shall specify DISTINCT or <ordered set function>.

6.3 <window function>

6) Let *WNS* be the <window name or specification>. Let *WDX* be a window structure descriptor that describes the window defined by *WNS*.

7) If <rank function type> or ROW_NUMBER is specified, then:

   a) If RANK or DENSE_RANK is specified, then the window ordering clause *WOC* of *WDX* shall be present.

   b) The window framing clause of *WDX* shall not be present.

   c) Case:

      i) If *WNS* is a <window name>, then let *WNS1* be *WNS*.

      ii) Otherwise, let *WNS1* be the <window specification details> contained in *WNS*.

   d) RANK( ) OVER *WNS* is equivalent to:

```
( COUNT (*) OVER ( WNS1 RANGE UNBOUNDED PRECEDING )
  - COUNT (*) OVER ( WNS1 RANGE CURRENT ROW ) + 1 )
```

   e) If DENSE_RANK is specified, then:

      i) Let VE_1, . . . , VE_N be an enumeration of the <value expression>s that are <sort key>s simply contained in *WOC*.

      ii) DENSE_RANK( ) OVER *WNS* is equivalent to the <window function>:

```
COUNT (DISTINCT ROW ( VE1, . . . , VEN ) )
OVER (WNS1 RANGE UNBOUNDED PRECEDING)
```

   f) ROW_NUMBER( ) OVER *WNS* is equivalent to the <window function>:

```
COUNT (*) OVER (WNS1 ROWS UNBOUNDED PRECEDING)
```

   g) Let *ANT1* be an approximate numeric type with implementation-defined precision. PERCENT_RANK( ) OVER *WNS* is equivalent to:

```
CASE
  WHEN COUNT(*) OVER (WNS1 RANGE BETWEEN UNBOUNDED PRECEDING
                                     AND UNBOUNDED FOLLOWING) = 1
    THEN CAST (0 AS ANT1)
    ELSE
         (CAST (RANK () OVER (WNS1) AS ANT1) - 1) /
         (COUNT (*) OVER (WNS1 RANGE BETWEEN UNBOUNDED PRECEDING
                                AND UNBOUNDED FOLLOWING) - 1)
END
```

   h) Let *ANT2* be an approximate numeric type with implementation-defined precision. CUME_DIST( ) OVER *WNS* is equivalent to:


```
( CAST ( COUNT (*) OVER
      ( WNS1 RANGE UNBOUNDED PRECEDING ) AS ANT2 ) /
    COUNT(*) OVER ( WNS1 RANGE BETWEEN UNBOUNDED PRECEDING
                                   AND UNBOUNDED FOLLOWING ) )
```

8) Let *SL* be the <select list> that simply contains *OF*.

NOTE 6 – If *OF* is originally contained in an <order by clause> of a cursor that is a simple table query, the syntactic transformation of Subclause 11.1, "<declare cursor>", must be applied prior to this rule.

9) Let *SQ* be the <set quantifier> of the <query specification> or <select statement: single row> that simply contains *SL*. If there is no <set quantifier>, then let *SQ* be a zero-length string.

10) If <in-line window specification> is specified, then:

a) Let *WS* be the <window specification>.

b) Let *WSN* be an implementation-dependent <window name> that is not equivalent to any other <window name> in the <table expression> or <select statement: single row> that simply contains *WS*.

c) Let *OFT* be the <window function type>.

d) Let *SLNEW* be the <select list> that is obtained from *SL* by replacing *OF* by:

```
OFT OVER WSN
```

e) Let *FC*, *WC*, *GBC*, and *HC* be <from clause>, <where clause>, <group by clause>, and <having clause>, respectively, of *TE*. If any of <where clause>, <group by clause>, or <having clause> is missing, then let *WC*, *GBC*, or *HC*, respectively, be a zero-length string.

f) Case:

i) If there is no <window clause> simply contained in *TE*, then let *WICNEW* be:

```
WINDOW WSN AS WS
```

ii) Otherwise, let *WIC* be the <window clause> simply contained in *TE* and let *WICNEW* be:

```
WIC, WSN AS WS
```

g) Let *TENEW* be:

```
FC WC GBC HC WICNEW
```

h) Case:

i) If *OF* is simply contained in a <query specification>, then that <query specification> is equivalent to:

```
SELECT SQ SLNEW TENEW
```

6.3 <window function>

ii) Otherwise, *OF* is simply contained in a <select statement: single row>. Let *STL* be the <select target list> of that <select statement: single row>. The <select statement: single row> is equivalent to:

```
SELECT SQ SLNEW INTO STL TENEW
```

Access Rules

None.

General Rules

1) The value of <window function> is the value of the <aggregate function>.

Conformance Rules

1) Without Feature T611, "Elementary OLAP operators", conforming SQL language shall not specify a <window function>.

2) Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify <window name>.

3) Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify PERCENT_RANK or CUME_DIST.

4) Without Feature T612, "Advanced OLAP operators", if ROW_NUMBER is specified, then the window ordering clause of *WDX* shall be present.

6.4 <value expression>

Function

Specify a value.

Format

```
<nonparenthesized value expression primary> ::=
        !! All alternatives from ISO/IEC 9075-2
      | <window function>
```

Syntax Rules

1) Replace SR 2) The declared type of a <value expression primary> is the declared type of the simply contained <unsigned value specification>, <column reference>, <set function specification>, <scalar subquery>, <case expression>, <value expression>, <cast specification>, <subtype treatment>, <attribute or method reference>, <reference resolution>, <collection value constructor>, <field reference>, <element reference>, <method invocation>, or <static method invocation>, or <window function>, or the effective returns type of the immediately contained <routine invocation>, respectively.

Access Rules

No additional Access Rules.

General Rules

1) Replace GR 5) The value of a <value expression primary> is the value of the simply contained <unsigned value specification>, <column reference>, <set function specification>, <rank function specification>, <scalar subquery>, <case expression>, <value expression>, <cast specification>, <subtype treatment>, <collection value constructor>, <field reference>, <element reference>, <method invocation>, <static method invocation>, <routine invocation>, <attribute or method reference>, or <window function>.

Conformance Rules

1) Insert this CR Without Feature T611, "Elementary OLAP operators", a <value expression> shall not be a <window function>.

(Blank page)

7 Query expressions

7.1 <table expression>

Format

```
<table expression> ::=
       <from clause>
       [ <where clause> ]
       [ <group by clause> ]
       [ <having clause> ]
       [ <window clause> ]
```

Syntax Rules

1) Replace SR 1) The result of a <table expression> is a derived table, whose row type *RT* is the row type of the result of the application of the last of the immediately contained <from clause>, <where clause>, <group by clause> or <having clause> specified in the <table expression>, together with the window structure descriptors defined by the <window clause>, if specified.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert this CR Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify a <window clause>.

7.2 <joined table>

Function

Specify a table derived from a Cartesian product, inner or outer join, or union join.

Format

!! *No additional Format items*

Syntax Rules

1) Replace SR 5)b) If a <value expression> directly contained in the <search condition> is a <set function specification>, then the <joined table> shall be contained in a <having clause> or <select list>, the <set function specification> shall contain an aggregated argument that is a column reference that is an outer reference, and every column reference contained in an aggregated argument of the <set function specification> shall be an outer reference.

2) Insert after SR 5) The <search condition> shall not contain a <window function> without an intervening <subquery>.

Access Rules

No additional Access Rules

General Rules

No additional General Rules

Conformance Rules

No additional Conformance Rules.

7.3 <where clause>

Function

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

Format

```
!! No additional Format items
```

Syntax Rules

1) Replace SR 2) If a <value expression> directly contained in the <search condition> is a <set function specification>, then the <where clause> shall be contained in a <having clause> or <select list>, the <set function specification> shall contain a column reference, and every column reference contained in an aggregated argument of the <set function specification> shall be an outer reference.

2) Insert after SR 2) The <search condition> shall not contain a <window function> without an intervening <subquery>.

3) Replace SR 4) No column reference contained in a <subquery> in the <search condition> that references a column of *T* shall be specified in an aggregated argument of a <set function specification>.

Access Rules

No additional Access Rules

General Rules

No additional General Rules

Conformance Rules

No additional Conformance Rules

7.4 <having clause>

Function

Specify a grouped table derived by the elimination of groups that do not satisfy a <search condition>.

Format

!! *No additional Format items*

Syntax Rules

1) Replace SR 4) Each column reference contained in a <subquery> in the <search condition> that references a column of *T* shall reference a column that is functionally dependent on *G* or shall be specified within an aggregated argument of a <set function specification>.

2) Insert after SR 4) The <search condition> shall not contain a <window function> without an intervening <subquery>.

Access Rules

No additional Access Rules

General Rules

1) Replace GR 2) When the <search condition> is applied to a given group of *R*, that group is the argument source of each <set function specification> directly contained in the <search condition>, unless the <column reference> in the <set function specification> is an outer reference.

Conformance Rules

1) Replace CR 2) Without Feature T301, "Functional dependencies", each column reference contained in a <subquery> in the <search condition> that references a column of *T* shall be one of the following:

   a) An unambiguous reference to a grouping column of *T*.

   b) Contained in an aggregated argument of a <set function specification>.

7.5 <window clause>

Function

Specify one or more window definitions.

Format

```
<window clause> ::=
       WINDOW <window definition list>

<window definition list> ::=
       <window definition> [ { <comma> <window definition> }... ]

<window definition> ::=
       <new window name> AS <window specification>

<new window name> ::= <window name>

<window specification> ::=
       <left paren> <window specification details> <right paren>

<window specification details> ::=
       [ <existing window name> ]
       [ <window partition clause> ]
       [ <window order clause> ]
       [ <window frame clause> ]

<existing window name> ::= <window name>

<window partition clause> ::=
       PARTITION BY <window partition column reference list>

<window partition column reference list> ::=
       <window partition column reference>
           [ { <comma> <window partition column reference> }... ]

<window partition column reference> ::=
       <column reference> [ <collate clause> ]

<window order clause> ::=
       ORDER BY <sort specification list>

<window frame clause> ::=
       <window frame units>
       <window frame extent>
       [ <window frame exclusion> ]

<window frame units> ::=
         ROWS
       | RANGE

<window frame extent> ::=
         <window frame start>
       | <window frame between>

<window frame start> ::=
         UNBOUNDED PRECEDING
       | <window frame preceding>
```

7.5 <window clause>

```
       | CURRENT ROW

<window frame preceding> ::=
       <unsigned value specification> PRECEDING

<window frame between> ::=
       BETWEEN <window frame bound 1>
           AND <window frame bound 2>

<window frame bound 1> ::=
       <window frame bound>

<window frame bound 2> ::=
       <window frame bound>

<window frame bound> ::=
         <window frame start>
       | UNBOUNDED FOLLOWING
       | <window frame following>

<window frame following> ::=
       <unsigned value specification> FOLLOWING

<window frame exclusion> ::=
         EXCLUDE CURRENT ROW
       | EXCLUDE GROUP
       | EXCLUDE TIES
       | EXCLUDE NO OTHERS
```

Syntax Rules

1) Let *TE* be the <table expression> that immediately contains the <window clause>.

2) <new window name> *NWN1* shall not be contained in the scope of another <new window name> *NWN2* such that *NWN1* and *NWN2* are equivalent.

3) Let *WDEF* be a <window definition>.

4) Each <column reference> contained in the <window partition clause> or <window order clause> of *WDEF* shall unambiguously reference a column of the derived table that is the result of *TE*. A column referenced in a <window partition clause> is a *partitioning column*.

   NOTE 7 – If *TE* is a grouped table, then the <column reference>s contained in <window partition clause> or <window order clause> must reference columns of the grouped table obtained by performing the syntactic transformation in Subclause 7.6, "<query specification>".

5) The declared type of a partitioning column shall not be LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

6) If *T* is a grouped table, then let *G* be the set of grouping columns of *T*. Each column reference contained in <window clause> that references a column of *T* shall reference a column that is functionally dependent on *G* or be contained in an aggregated argument of a <set function specification>.

7) A <window clause> shall not contain a <window function> without an intervening <subquery>.

8) If *WDEF* specifies <window frame between>, then:

   a) <window frame bound 1> shall not specify UNBOUNDED FOLLOWING.

   b) <window frame bound 2> shall not specify UNBOUNDED PRECEDING.

   c) If <window frame bound 1> specifies CURRENT ROW, then <window frame bound 2> shall not specify <window frame preceding>.

   d) If <window frame bound 1> specifies <window frame following>, then <window frame bound 2> shall not specify <window frame preceding> or CURRENT ROW.

9) If *WDEF* specifies <window frame extent>, and does not specify <window frame between>, then let *WAGS* be the <window frame start>. The <window frame extent> is equivalent to

```
BETWEEN WAGS AND CURRENT ROW
```

10) If *WDEF* specifies an <existing window name> *EWN*, then:

   a) *WDEF* shall be within the scope of a <window name> that is equivalent to <existing window name>.

   b) Let *WDX* be the window structure descriptor identified by *EWN*.

   c) *WDEF* shall not specify <window partition clause>.

   d) If *WDX* has a window ordering clause, then *WDEF* shall not specify <window order clause>.

   e) *WDX* shall not have a window framing clause.

11) If *WDEF*'s <window frame clause> specifies <window frame preceding> or <window frame following>, then let *UVS* be the <unsigned value specification> simply contained in the <window frame preceding> or <window frame following>.

   Case:

   a) If RANGE is specified, then *WDEF*'s <window order clause> shall contain a single <sort key> *SK*. The declared type of *SK* shall be numeric, datetime, or interval. The declared type of *UVS* shall be numeric if the declared type of *SK* is numeric; otherwise, it shall be an interval type that may be added to or subtracted from the declared type of *SK* according to the Syntax Rules of Subclause 6.28, "<datetime value expression>", and Subclause 6.29, "<interval value expression>", in ISO/IEC 9075-2.

   b) If ROWS is specified, then the declared type of *UVS* shall be exact numeric with scale 0 (zero).

12) The scope of the <new window name> simply contained in *WDEF* consists of any <window definition>s that follow *WDEF* in the <window clause>, together with the <select list> of the <query specification> or <select statement: single row> that simply contains the <window clause>. If the <window clause> is simply contained in a <query specification> that is the <query expression body> of a <declare cursor> that is a simple table query, then the scope of <new window name> also includes the <order by clause> of the <declare cursor>.

7.5 <window clause>

13) Two window structure descriptors *WD1* and *WD2* are *order-equivalent* if all of the following conditions are met:

a) Let $WPCR1_i$, 1 (one) $\leq i \leq N1$, and $WPCR2_i$, 1 (one) $\leq i \leq N2$, be enumerations of the <window partition column reference>s contained in the window partitioning clauses of *WD1* and *WD2*, respectively, in order from left to right. $N1 = N2$, and, for all i, $WPCR1_i$ and $WPCR2_i$ are equivalent column references.

b) Let $SS1_i$, 1 (one) $\leq i \leq M1$, and $SS2_i$, 1 (one) $\leq i \leq M2$, be enumerations of the <sort specification>s contained in the window ordering clauses of *WD1* and *WD2*, respectively, in order from left to right. $M1 = M2$, and, for all i, $SS1_i$ and $SS2_i$ contain <sort key>s that are equivalent column references, specify or imply the same <ordering specification>, specify or imply the same <collate clause>, if any, and specify or imply the same <null ordering>.

Access Rules

None.

General Rules

1) Let *TE* be the <table expression> that simply contains the <window clause>. Let *SL* be the <select list> of the <query specification> or <select statement: single row> that immediately contains *TE*.

Case:

a) If *SL* does not simply contain a <window function>, then the <window clause> is disregarded, and the result of *TE* is the result of the last <from clause>, <where clause>, <group by clause> or <having clause> of *TE*.

b) Otherwise, let *RTE* be the result of the last <from clause> or <where clause> simply contained in *TE*.

NOTE 8 – Although it is permissible to have a <group by clause> or a <having clause> with a <window clause>, if there are any <window function>s, then the <group by clause> and <having clause> are removed by a syntactic transformation in Subclause 7.6, "<query specification>", and so are not considered here.

i) A window structure descriptor *WDESC* is created for each <window definition> *WDEF*, as follows:

1) *WDESC*'s window name is the <new window name> simply contained in *WDEF*.

2) If <existing window name> is specified, then let *EWN* be the <existing window name> simply contained in *WDEF* and let *WDX* be the window structure descriptor identified by *EWN*.

3) If <existing window name> is specified and the window ordering clause of *WDX* is present, then the ordering window name of *WDESC* is *EWN*; otherwise, there is no ordering window name.

4) Case:

A) If *WDEF* simply contains <window partition clause> *WDEFWPC*, then *WDESC*'s window partitioning clause is *WDEFWPC*.

B) If <existing window name> is specified, then *WDESC*'s window partitioning clause is the window partitioning clause of *WDX*.

C) Otherwise, *WDESC* has no window partitioning clause.

5) Case:

A) If *WDEF* simply contains <window order clause> *WDEFWOC*, then *WDESC*'s window ordering clause is *WDEFWOC*.

B) If <existing window name> is specified, then *WDESC*'s window ordering clause is the window ordering clause of *WDX*.

C) Otherwise, *WDESC* has no window ordering clause.

6) If *WDEF* simply contains <window frame clause> *WDEFWFC*, then *WDESC*'s window framing clause is *WDEFWFC*; otherwise, *WDESC* has no windows framing.

ii) The result of <window clause> is *RTE*, together with the window structure descriptors defined by the <window clause>.

2) Let *WD* be a window structure descriptor.

3) *WD* defines, for each row *R* of *RTE*, the *window partition* of *R* under *WD*, consisting of the multiset of rows of *RTE* that are not distinct from *R* in the window partitioning columns of *WD*. If *WD* has no window partitioning clause, then the window partition of *R* is the entire result *RTE*.

4) *WD* also defines the window ordering of the rows of each window partition defined by *WD*, according to the General Rules of Subclause 8.2, "<sort specification list>", using the <sort specification list> simply contained in *WD*'s window ordering clause. If *WD* has no window ordering clause, then the window ordering is entirely implementation-dependent, and all rows are peers. Although the window ordering of peer rows within a window partition is implementation-dependent, the window ordering shall be the same for all window structure descriptors that are order-equivalent. It shall also be the same for any pair of windows *W1* and *W2* such that *W1* is the ordering window for *W2*.

5) *WD* also defines for each row *R* of *RTE* the window frame *WF* of *R*, consisting of a multiset of rows. *WF* is defined as follows:

a) If *WD* has no window framing clause, then

Case:

i) If the window ordering clause of *WD* is not present, then *WF* is the window partition of *R*.

ii) Otherwise, *WF* consists of all rows of the window partition of *R* that precede *R* or are peers of *R* in the window ordering of the window partition defined by the window ordering clause.

7.5 <window clause>

b) Otherwise, let *WF* initially be the window partition of *R* defined by *WD*. Let *WFC* be the window framing clause of *WD*. Let *WFB1* be the <window frame bound 1> and let *WFB2* be the <window frame bound 2> contained in *WFC*.

i) If RANGE is specified, then:

1) In the following subrules, when performing addition or subtraction to combine a datetime and a year-month interval, if the result would raise the exception condition *data exception — datetime field overflow* because the <primary datetime field> DAY is not valid for the computed value of the <primary datetime field>s YEAR and MONTH, then the <primary datetime field> DAY is set to the last day that is valid for the <primary datetime field>s YEAR and MONTH, and no exception condition is raised.

2) Case:

NOTE 9 – In the following subrules, if *WFB1* specifies UNBOUNDED PRECEDING, then no rows are removed from *WF* by this step. *WFB1* may not be UNBOUNDED FOLLOWING.

A) If *WFB1* specifies <window frame preceding>, then let *V1P* be the value of the <unsigned value specification>.

I) If *V1P* is negative or the null value, then an exception condition is raised: *data exception — invalid preceding or following size in window function*.

II) Otherwise, let *SK* be the only <sort key> contained in the window ordering clause of *WD*. Let *VSK* be the value of *SK* for the current row.

Case:

1) If *VSK* is the null value and if NULLS LAST is specified or implied, then remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is not the null value.

2) If *VSK* is not the null value, then:

a) If NULLS FIRST is specified or implied, then remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is the null value.

b) Case:

i) If the <ordering specification> contained in the window ordering clause specifies DESC, then let *BOUND* be the value *VSK+V1P*. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is greater than *BOUND*.

ii) Otherwise, let *BOUND* be the value *VSK–V1P*. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is less than *BOUND*.

B) If *WFB1* specifies CURRENT ROW, then remove from *WF* all rows that are not peers of the current row and that precede the current row in the window ordering defined by *WD*.

C) If *WFB1* specifies <window frame following>, then let *V1F* be the value of the <unsigned value specification>.

I) If *V1F* is negative or the null value, then an exception condition is raised: *data exception — invalid preceding or following size in window function*.

II) Otherwise, let *SK* be the only <sort key> contained in the window ordering clause of *WD*. Let *VSK* be the value of *SK* for the current row.

Case:

1) If *VSK* is the null value and if NULLS LAST is specified or implied, then remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is not the null value.

2) If *VSK* is not the null value, then:

a) If NULLS FIRST is specified or implied, then remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is the null value.

b) Case:

i) If the <ordering specification> contained in the window ordering clause specifies DESC, then let *BOUND* be the value *VSK–V1F*. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is greater than *BOUND*.

ii) Otherwise, let *BOUND* be the value *VSK+V1F*. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is less than *BOUND*.

3) Case:

NOTE 10 – In the following subrules, if *WFB2* specifies UNBOUNDED FOLLOWING, then no rows are removed from *WF* by this step. *WFB2* may not be UNBOUNDED PRECEDING.

A) If *WFB2* specifies <window frame preceding>, then let *V2P* be the value of the <unsigned value specification>.

I) If *V2P* is negative or the null value, then an exception condition is raised: *data exception — invalid preceding or following size in window function*.

II) Otherwise, let *SK* be the only <sort key> contained in the window ordering clause of *WD*. Let *VSK* be the value of *SK* for the current row.

Case:

1) If *VSK* is the null value and if NULLS FIRST is specified or implied, then remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is not the null value.

2) If *VSK* is not the null value, then:

a) If NULLS LAST is specified or implied, then remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is the null value.

7.5 <window clause>

b) Case:

i) If the <ordering specification> contained in the window ordering clause specifies DESC, then let *BOUND* be the value *VSK+V2P*. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is less than *BOUND*.

ii) Otherwise, let *BOUND* be the value *VSK–V2P*. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is greater than *BOUND*.

B) If *WFB2* specifies CURRENT ROW, then remove from *WF* all rows following the current row in the ordering defined by *WD* that are not peers of the current row.

C) If *WFB2* specifies <window frame following>, then let *V2F* be the value of the <unsigned value specification>.

I) If *V2F* is negative or the null value, then an exception condition is raised: *data exception — invalid preceding or following size in window function*.

II) Otherwise, let *SK* be the only <sort key> contained in the window ordering clause of *WD*. Let *VSK* be the value of *SK* for the current row.

Case:

1) If *VSK* is the null value and if NULLS FIRST is specified or implied, then remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is not the null value.

2) If *VSK* is not the null value, then:

a) If NULLS LAST is specified or implied, then remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is the null value.

b) Case:

i) If the <ordering specification> contained in the <window order clause> specifies DESC, then let *BOUND* be the value *VSK–V2F*. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is less than *BOUND*.

ii) Otherwise, let *BOUND* be the value *VSK+V2F*. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is greater than *BOUND*.

ii) If ROWS is specified, then:

1) Case:

NOTE 11 – In the following subrules, if *WFB1* specifies UNBOUNDED PRECEDING, then no rows are removed from *WF* by this step. *WFB1* may not be UNBOUNDED FOLLOWING.

A) If *WFB1* specifies <window frame preceding>, then let *V1P* be the value of the <unsigned value specification>.

I) If *V1P* is negative or the null value, then an exception condition is raised:

data exception — invalid preceding or following size in window function.

II) Otherwise, remove from *WF* all rows that are more than *V1P* rows preceding the current row in the window ordering defined by *WD*.

B) If *WFB1* specifies CURRENT ROW, then remove from *WF* all rows that precede the current row in the window ordering defined by *WD*.

NOTE 12 – This step removes any peers of the current row that precede it in the implementation-dependent window ordering.

C) If *WFB1* specifies <window frame following>, then let *V1F* be the value of the <unsigned value specification>.

I) If *V1F* is negative or the null value, then an exception condition is raised: *data exception — invalid preceding or following size in window function*.

II) Otherwise, remove from *WF* all rows that precede the current row and all rows that are less than *V1F* rows following the current row in the window ordering defined by *WD*.

NOTE 13 – If *V1F* is zero, then the current row is not removed from *WF* by this step; otherwise, the current row is removed from *WF*.

2) Case:

NOTE 14 – In the following subrules, if *WFB2* specifies UNBOUNDED FOLLOWING, then no rows are removed from *WF* by this step. *WFB2* may not be UNBOUNDED PRECEDING.

A) If *WFB2* specifies <window frame preceding>, then let *V2P* be the value of the <unsigned value specification>.

I) If *V2P* is negative or the null value, then an exception condition is raised: *data exception — invalid preceding or following size in window function*.

II) Otherwise, remove from *WF* all rows that follow the current row and all rows that are less than *V2P* rows preceding the current row in the window ordering defined by *WD*.

NOTE 15 – If *V2P* is zero, then the current row is not removed from *WF* by this step; otherwise, the current row is removed from *WF*.

B) If *WFB2* specifies CURRENT ROW, then remove from *WF* all rows that follow the current row in the window ordering defined by *WD*.

NOTE 16 – This step removes any peers of the current row that follow it in the implementation-dependent window ordering.

C) If *WFB2* specifies <window frame following>, then let *V2F* be the value of the <unsigned value specification>.

I) If *V2F* is negative or the null value, then an exception condition is raised: *data exception — invalid preceding or following size in window function*.

II) Otherwise, remove from *WF* all rows that are more than *V2F* rows following the current row in the window ordering defined by *WD*.

iii) If <window framing exclusion> *WFE* is specified, then

7.5 <window clause>

Case:

1) If EXCLUDE CURRENT ROW is specified and the current row is still a member of *WF*, then remove the current row from *WF*.

2) If EXCLUDE GROUP is specified, then remove the current row and any peers of the current row from *WF*.

3) If EXCLUDE TIES is specified, then remove any rows other than the current row that are peers of the current row from *WF*.

NOTE 17 – If the current row is already removed from *WF*, then it remains removed from *WF*.

NOTE 18 – If EXCLUDE NO OTHERS is specified, then no additional rows are removed from *WF* by this Rule.

Conformance Rules

1) Without Feature T611, "Elementary OLAP operators", conforming SQL language shall not specify <window specification>.

2) Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify a <window clause>.

3) Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify <existing window name>.

4) Without Feature T301, "Functional dependencies", if *T* is a grouped table, then each column reference contained in <window clause> that references a column of *T* shall be a reference to a grouping column of *T* or be contained in an aggregated argument of a <set function specification>.

5) Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify <window framing exclusion>.

6) Without Feature S024, "Enhanced structured types", the declared type of a partitioning column shall not be ST-ordered.

7.6 <query specification>

Function

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

Format

```
!! No additional Format items.
```

Syntax Rules

1) Insert after SR 10) Each column reference contained in a <window function> shall unambiguously reference a column of *T*.

2) Insert this SR If both of the following two conditions are satisfied, then *QS* is a *grouped, windowed query*:

   a) *T* is a grouped table.

   b) Some <derived column> simply contained in *QS* simply contains a <window function>.

3) Insert this SR A grouped, windowed query *GWQ* is transformed to an equivalent <query specification> as follows:

   a) If *GWQ* contains an <in-line window specification>, then apply the syntactic transformation specified in Subclause 6.3, "<window function>".

   b) If the <select list> of *GWQ* contains <asterisk> or <qualified asterisk>, then apply the syntactic transformations specified in Subclause 7.11, "<query specification>", in ISO/IEC 9075-2.

   c) Let *GWQ2* be the result of the preceding transformations, if any.

   d) Let *SL*, *FC*, *WC*, *GBC*, *HC*, and *WIC* be the <select list>, <from clause>, <where clause>, <group by clause>, <having clause>, and <window clause>, respectively, of *GWQ2*. If any of <where clause>, <group by clause>, or <having clause>, are missing, then let *WC*, *GBC*, and *HC*, respectively, be a zero-length string. Let *SQ* be the <set quantifier> immediately contained in the <query specification> of *GWQ2*, if any; otherwise, let *SQ* be a zero-length string.

      NOTE 19 – *GWQ2* can not lack a <window clause>, since the syntactic transformation of Subclause 6.3, "<window function>", will create one if there is not one in *GWQ* already.

   e) Let *N1* be the number of <set function specification>s simply contained in *GWQ2*.

   f) Let SFS_i, 1 (one) $\leq i \leq N1$, be an enumeration of the <set function specification>s simply contained in *GWQ2*.

   g) Let $SFSI_i$, 1 (one) $\leq i \leq N1$, be a list of <identifier>s that are distinct from each other and distinct from all <identifier>s contained in *GWQ2*.

   h) If *N1* = 0 (zero), then let *SFSL* be a zero-length string; otherwise, let *SFSL* be:

      SFS_1 AS $SFSI_1$, SFS_2 AS $SFSI_2$, . . . , SFS_{N1} AS $SFSI_{N1}$

7.6 <query specification>

i) Let *HCNEW* be obtained from *HC* by replacing each <set function specification> SFS_i by the corresponding <identifier> $SFSI_i$.

j) Let *N2* be the number of <column reference>s that are contained in *SL* or *WIC* without an intervening <subquery> or <set function specification>.

k) Let CR_j, 1 (one) $\leq j \leq$ *N2*, be an enumeration of the <column reference>s that are contained in *SL* or *WIC* without an intervening <subquery> or <set function specification>.

l) Let CRI_j, 1 (one) $\leq j \leq$ *N2*, be a list of <identifier>s that are distinct from each other, distinct from all identifiers in *GWQ2*, and distinct from all $SFSI_i$.

m) If *N2* = 0 (zero), then let *SFSL* be a zero-length string; otherwise, let *CRL* be:

CR_1 AS CRI_2, CR_2 AS CRI_2, . . . , CR_{N2} AS CRI_{N2}

n) Let *N3* be the number of <derived column>s simply contained in *SL* that do not specify <as clause>.

o) Let $DCOL_k$, 1 (one) $\leq k \leq$ *N3*, be the <derived column>s simply contained in *SL* that do not specify an <as clause>. For each *k*, let $COLN_k$ be the <column name> determined as follows:

i) If $DCOL_k$ is a single column reference, then let $COLN_k$ be the <column name> of the column designated by the column reference.

ii) Otherwise, let $COLN_k$ be an implementation-dependent <column name> that is not equivalent to the <column name> of any column, other than itself, of a table referenced by any <table reference> contained in the SQL-statement.

p) Let *SL2* be obtained from *SL* by replacing each <derived column> $DCOL_k$ by

$DCOL_k$ AS $COLN_k$

q) Let *GWQN* be an arbitrary <identifier>.

r) Let *SLNEW* be the <select list> obtained from *SL2* by replacing each simply contained <set function specification> SFS_i by $GWQN.SFSI_i$ and replacing each <column reference> CR_j that is contained without an intervening <subquery> or <set function specification> by $GWQN.CRI_j$.

s) Let *WICNEW* be the <window clause> obtained from *WIC* by replacing each <set function specification> SFS_i by $GWQN.SFSI_i$ and by replacing each <column reference> CR_j by $GWQN.CRI_j$.

t) If either *SFSL* or *CRL* is a zero-length string, then let *COMMA* be a zero-length string; otherwise, let *COMMA* be "," (a <comma>).

u) *GWQ* is equivalent to the following <query specification>:


```
SELECT SLNEW
FROM ( SELECT SQ SFSL COMMA CRL
       FC
       WC
       GBC
       HC ) AS GWQN
WICNEW
```

4) Insert after SR 11)c) The <query specification> contains a <window function> that specifies ROW_NUMBER or whose associated <window specification> specifies ROWS.

5) Replace SR 12) If <table expression> does not immediately contain a <group by clause> and <select list> contains either a <value expression> that contains a <set function specification> that has an aggregated argument that contains a reference to a column of *T* or a <value expression> that directly contains a <set function specification> that does not have an aggregated argument that contains an outer reference, then GROUP BY ( )is implicit.

6) Replace SR 13) If *T* is a grouped table, then let *G* be the set of grouping columns of *T*. In each <value expression> contained in <select list> , each column reference that references a column of *T* shall reference some column *C* that is functionally dependent on *G* or shall be contained in an aggregated argument of a <set function specification>.

7) Insert after SR 16)a)viii) A <window function> whose <window function type> does not contain <rank function type>, ROW_NUMBER, or an <aggregate function> that simply contains COUNT.

Access Rules

No additional Access Rules.

General Rules

1) Replace introduction of GR 1)b)ii) If *T* has one or more groups, then each <value expression> is applied to each group of *T* yielding a table *TEMP* of *M* rows, where *M* is the number of groups in *T*. The *i*-th column of *TEMP* contains the values derived by the evaluation of the *i*-th <value expression>. When a <value expression> is applied to a given group of *T*, that group is the argument source of each <set function specification> in the <value expression>.

Conformance Rules

1) Replace CR 3) Without Feature T301, "Functional dependencies", if *T* is a grouped table, then in each <value expression>, each column reference that references a column of *T* shall reference a grouping column or be specified in an aggregated argument of a <set function specification>.

(Blank page)

8 Additional common elements

8.1 <aggregate function>

Function

Specify a value computed from a multiset of rows.

Format

```
<aggregate function> ::=
        COUNT <left paren> <asterisk> <right paren> [ <filter clause> ]
      | <general set function> [ <filter clause> ]
      | <binary set function> [ <filter clause> ]
      | <ordered set function> [ <filter clause> ]

<general set function> ::=
      <set function type> <left paren> [ <set quantifier> ]
          <value expression> <right paren>

<set function type> ::=
      <computational operation>

<computational operation> ::=
        AVG | MAX | MIN | SUM
      | EVERY | ANY | SOME
      | COUNT
      | STDDEV_POP | STDDEV_SAMP | VAR_SAMP | VAR_POP

<set quantifier> ::=
        DISTINCT
      | ALL

<filter clause> ::=
      FILTER <left paren> WHERE <search condition> <right paren>

<binary set function> ::=
      <binary set function type> <left paren>
        <dependent variable expression> <comma>
        <independent variable expression> <right paren>

<binary set function type> ::=
        COVAR_POP | COVAR_SAMP | CORR | REGR_SLOPE
      | REGR_INTERCEPT | REGR_COUNT | REGR_R2 | REGR_AVGX | REGR_AVGY
      | REGR_SXX | REGR_SYY | REGR_SXY

<dependent variable expression> ::=
      <numeric value expression>

<independent variable expression> ::=
      <numeric value expression>

<ordered set function> ::=
        <hypothetical set function>
```

8.1 <aggregate function>

```
      | <inverse distribution function>

<hypothetical set function> ::=
      <rank function type> <left paren>
        <hypothetical set function value expression list> <right paren>
      <within group specification>

<within group specification> ::=
      WITHIN GROUP <left paren> ORDER BY
        <sort specification list> <right paren>

<hypothetical set function value expression list> ::=
      <value expression> [ { <comma> <value expression> }... ]

<inverse distribution function> ::=
      <inverse distribution function type> <left paren>
        <inverse distribution function argument> <right paren>
      <within group specification>

<inverse distribution function argument> ::=
      <numeric value expression>

<inverse distribution function type> ::=
        PERCENTILE_CONT
      | PERCENTILE_DISC
```

Syntax Rules

1) Let *AF* be the <aggregate function>.

2) If STDDEV_POP, STDDEV_SAMP, VAR_POP or VAR_SAMP is specified, then <set quantifier> shall not be specified.

3) In a <general set function>, if <set quantifier> is not specified, then ALL is implicit.

4) The argument source of an <aggregate function> is

   Case:

   a) If *AF* is immediately contained in a <set function specification>, then a table or group of a grouped table as specified in Subclause 7.4, "<having clause>", and Subclause 7.6, "<query specification>".

   b) Otherwise, the multiset of rows in the current row's window frame defined by the window structure descriptor identified by the <window aggregate> that simply contains *AF*, as defined in Subclause 7.5, "<window clause>".

5) Let *T* be the argument source of *AF*.

6) If COUNT is specified, then the declared type of the result is exact numeric with implementation-defined precision and scale of 0 (zero).

7) If <general set function> is specified, then:

   a) The <value expression> shall not contain a <window function>.

   b) Let *DT* be the declared type of the <value expression>.

c) If *AF* specifies a <general set function> whose <set quantifier> is DISTINCT, then *DT* shall not be LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

d) If *AF* specifies a <set function type> that is MAX or MIN, then *DT* shall not be LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

e) If EVERY, ANY, or SOME is specified, then *DT* shall be boolean and the declared type of the result is boolean.

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

g) If SUM or AVG is specified, then:

i) *DT* shall be a numeric type or an interval type.

ii) If SUM is specified and *DT* is exact numeric with scale *S*, then the declared type of the result is exact numeric with implementation-defined precision and scale *S*.

iii) If AVG is specified and *DT* is exact numeric, then the declared type of the result is exact numeric with implementation-defined precision not less than the precision of *DT* and implementation-defined scale not less than the scale of *DT*.

iv) If *DT* is approximate numeric, then the declared type of the result is approximate numeric with implementation-defined precision not less than the precision of *DT*.

v) If *DT* is interval, then the declared type of the result is interval with the same precision as *DT*.

h) If VAR_POP or VAR_SAMP is specified, then the declared type of the result is approximate numeric with implementation-defined precision not less than the precision of *DT*.

i) `STDDEV_POP(`*X*`)` is equivalent to `SQRT(VAR_POP(`*X*`))`.

j) `STDDEV_SAMP(`*X*`)` is equivalent to `SQRT(VAR_SAMP(`*X*`))`.

k) If the declared type of the result is character string, then the collating sequence and the coercibility characteristic are determined as in Subclause 4.2.3, "Rules determining collating sequence usage", in ISO/IEC 9075-2:1999.

8) A <filter clause> shall not contain a <subquery>, a <window function>, or an outer reference.

9) If <binary set function> is specified, then:

a) The <dependent variable expression> *DVE* and the <independent variable expression> *IVE* shall not contain a <window function>.

b) Let *DTDVE* be the declared type of *DVE* and let *DTIVE* be the declared type of *IVE*.

c) Case:

i) The declared type of REGR_COUNT is exact numeric with implementation-defined precision and scale of 0 (zero).

ii) Otherwise, the declared type of the result is approximate numeric with implementation-defined precision not less than the precision of the declared type of *DVE* and not less than the precision of the declared type of *IVE*.

8.1 <aggregate function>

10) If <hypothetical set function> is specified, then:

a) The <hypothetical set function> shall not contain a <window function>, a <set function specification>, or a <subquery>.

b) If a <value expression> simply contained in the <hypothetical set function value expression list> or the <sort specification list> contains a column reference that is an outer reference, then that outer reference shall be the only column reference contained in the <value expression>.

c) The number of <value expression>s simply contained in <hypothetical set function value expression list> shall be the same as the number of <sort key>s simply contained in the <sort specification list>.

d) The declared type of each <value expression> simply contained in the <hypothetical set function value expression list> shall be the same as the declared type of the corresponding <sort key> simply contained in the <sort specification list>, with the same precision and scale, if applicable. If this is a character type, then the character repertoires shall be the same, and the coercibility of the <value expression> shall not be Explicit. The collation is determined by Table 1, "Collating coercibility rules for monadic operators", in ISO/IEC 9075-2, using the coercibility and collation of the <sort key>.

e) Case:

i) If RANK or DENSE_RANK is specified, then the declared type of the result is exact numeric with implementation-defined precision and with scale 0 (zero).

ii) Otherwise, the declared type of the result is approximate numeric with implementation-defined precision.

11) If <inverse distribution function> is specified, then:

a) The <within group specification> shall contain a single <sort specification>.

b) The <inverse distribution function> shall not contain a <window function>, a <set function specification>, or a <subquery>.

c) If the <inverse distribution function argument> or the <sort specification> contains a column reference that is an outer reference, then that outer reference shall be the only column reference contained in the <inverse distribution function argument> or <sort specification>.

d) Let *DT* be the declared type of the <value expression> simply contained in the <sort specification>.

e) If PERCENTILE_CONT is specified, then *DT* shall be numeric or interval.

f) The declared type of the result is

Case:

i) If *DT* is numeric, then approximate numeric with implementation-defined precision.

ii) If *DT* is interval, then *DT*.

Access Rules

None.

General Rules

1) Case:

   a) If <filter clause> is specified, then the <search condition> is applied to each row of *T*. Let *T1* be the multiset of rows of *T* for which the result of the <search condition> is true.

   b) Otherwise, let *T1* be *T*.

2) If COUNT(*) is specified, then the result is the cardinality of *T1*.

3) If <general set function> is specified, then:

   a) Let *TX* be the single-column table that is the result of applying the <value expression> to each row of *T1* and eliminating null values. If one or more null values are eliminated, then a completion condition is raised: *warning — null value eliminated in set function*.

   b) Case:

      i) If DISTINCT is specified, then let *TXA* be the result of eliminating redundant duplicate values from *TX*, using the comparison rules specified in Subclause 8.2, "<comparison predicate>", in ISO/IEC 9075-2, to identify the redundant duplicate values.

      ii) Otherwise, let *TXA* be *TX*.

   c) Let *N* be the cardinality of *TXA*.

   d) Case:

      i) If the <general set function> COUNT is specified, then the result is *N*.

      ii) If *TXA* is empty, then the result is the null value.

      iii) If AVG is specified, then the result is the average of the values in *TXA*.

      iv) If MAX or MIN is specified, then the result is respectively the maximum or minimum value in *TXA*. These results are determined using the comparison rules specified in Subclause 8.2, "<comparison predicate>", in ISO/IEC 9075-2. If *DT* is a user-defined type and the comparison of two values in *TXA* results in *unknown*, then the maximum or minimum of *TXA* is implementation-dependent.

      v) If SUM is specified, then the result is the sum of the values in *TXA*. If the sum is not within the range of the declared type of the result, then an exception condition is raised: *data exception — numeric value out of range*.

      vi) If EVERY is specified, then

      Case:

      1) If the value of some element of TXA is *false*, then the result is *false*.

      2) Otherwise, the result is *true*.

8.1 <aggregate function>

vii) If ANY or SOME is specified, then

Case:

1) If the value of some element of *TXA* is *true* , then the result is *true* .

2) Otherwise, the result is *false* .

viii) If VAR_POP or VAR_SAMP is specified, then let *S1* be the sum of values in the column of *TXA*, and *S2* be the sum of the squares of the values in the column of *TXA*.

1) If VAR_POP is specified, then the result is (*S2-S1*S1/N*)/*N*.

2) If VAR_SAMP is specified, then:

A) If *N* is 1 (one), then the result is the null value.

B) Otherwise, the result is (*S2-S1*S1/N*)/(*N*-1).

4) If <binary set function type> is specified, then:

a) Let *TXA* be the two-column table that is the result of applying the <dependent variable expression> and the <independent variable expression> to each row of *T1* and eliminating each row in which either <dependent variable expression> or <independent variable expression> is the null value. If one or more null values are eliminated, then a completion condition is raised: *warning — null value eliminated in set function*.

b) Let *N* be the cardinality of *TXA*, let *SUMX* be the sum of the column of values of <independent variable expression>, let *SUMY* be the sum of the column of values of <dependent variable expression>, let *SUMX2* be the sum of the squares of values in the <independent variable expression> column, let *SUMY2* be the sum of the squares of values in the <dependent variable expression> column, and let *SUMXY* be the sum of the row-wise products of the value in the <independent variable expression> column times the value in the <dependent variable expression> column.

c) Case:

i) If REGR_COUNT is specified, then the result is *N*.

ii) If *N* is 0 (zero), then the result is the null value.

iii) If REGR_SXX is specified, then the result is (*SUMX2-SUMX*SUMX/N*).

iv) If REGR_SYY is specified, then the result is (*SUMY2-SUMY*SUMY/N*).

v) If REGR_SXY is specified, then the result is (*SUMXY-SUMX*SUMY/N*).

vi) If REGR_AVGX is specified, then the result is *SUMX/N*.

vii) If REGR_AVGY is specified, then the result is *SUMY/N*.

viii) If COVAR_POP is specified, then the result is (*SUMXY-SUMX*SUMY/N*)/*N*.

ix) If COVAR_SAMP is specified, then

Case:

1) If *N* is 1 (one), then the result is the null value.

2) Otherwise, the result is (*SUMXY*-*SUMX***SUMY*/*N*)/(*N*-1)

x) If CORR is specified, then

Case:

1) If *N***SUMX2* equals *SUMX***SUMX*, then the result is the null value.

NOTE 20 – In this case, all remaining values of <independent variable expression> are equal, and consequently the <independent variable expression> does not correlate with the <dependent variable expression>.

2) If *N***SUMY2* equals *SUMY***SUMY*, then the result is the null value.

NOTE 21 – In this case, all remaining values of <dependent variable expression> are equal, and consequently the <dependent variable expression> does not correlate with the <independent variable expression>.

3) Otherwise, the result is SQRT(POWER(*N***SUMXY*-*SUMX***SUMY*,2) / ((*N***SUMX2*-*SUMX***SUMX*)*(*N***SUMY2*-*SUMY***SUMY*))). If the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result data type, then the result is the null value.

xi) If REGR_R2 is specified, then

Case:

1) If *N***SUMX2* equals *SUMX***SUMX*, then the result is the null value.

NOTE 22 – In this case, all remaining values of <independent variable expression> are equal, and consequently the least-squares fit line would be vertical, or, if *N* = 1 (one), there is no uniquely determined least-squares-fit line.

2) If *N***SUMY2* equals *SUMY***SUMY*, then the result is 1 (one).

NOTE 23 – In this case, all remaining values of <dependent variable expression> are equal, and consequently the least-squares fit line is horizontal.

3) Otherwise, the result is POWER(*N***SUMXY*-*SUMX***SUMY*,2) / ((*N***SUMX2*-*SUMX***SUMX*)*(*N***SUMY2*-*SUMY***SUMY*)). If the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result data type, then the result is the null value.

xii) If REGR_SLOPE(*Y*, *X*) is specified, then

Case:

1) If *N***SUMX2* equals *SUMX***SUMX*, then the result is the null value.

NOTE 24 – In this case, all remaining values of <independent variable expression> are equal, and consequently the least-squares fit line would be vertical, or, if *N* = 1 (one), then there is no uniquely determined least-squares-fit line.

2) Otherwise, the result is (*N***SUMXY*-*SUMX***SUMY*) / (*N***SUMX2*-*SUMX***SUMX*). If the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result data type, then the result is the null value.

xiii) If REGR_INTERCEPT is specified, then

8.1 <aggregate function>

Case:

1) If *N*SUMX2* equals *SUMX*SUMX*, then the result is the null value.

   NOTE 25 – In this caes, all remaining values of <independent variable expression> are equal, and consequently the least-squares fit line would be vertical, or, if N = 1 (one), then there is no uniquely determined least-squares-fit line.

2) Otherwise, the result is (*SUMY*SUMX2-SUMX*SUMXY*) / (*N*SUMX2-SUMX*SUMX*). If the exponent of the approximate mathematical result of the operation is not within the implementation-defined exponent range for the result data type, then the result is the null value.

5) If <hypothetical set function> is specified, then

   a) Let *WIFT* be the <rank function type>.

   b) Let *TNAME* be an implementation-dependent name for *T1*.

   c) Let *K* be the number of <value expression>s simply contained in <hypothetical set function value expression list>.

   d) Let VE_1, . . . , VE_K be the <value expression>s simply contained in the <hypothetical set function value expression list>.

   e) Let *WIFTVAL*, *MARKER* and CN_1, . . . , CN_K be distinct implementation-dependent column names.

   f) Let SP_1, . . . , SP_K be the <sort specification>s simply contained in the <sort specification list>. For each i, let WSP_i be the <sort specification> obtained from SP_i by replacing the <sort key> with CN_i.

   g) The result is the result of the <scalar subquery>

```
( SELECT WIFTVAL
    FROM ( SELECT MARKER, WIFT() OVER
                ( ORDER BY WSP1,  . . .  , WSPK )
             FROM ( SELECT 0, SK1,  . . .  , SKK
                    FROM TNAME
                      UNION
                        VALUES ( 1, VE1,  . . .  , VEK ) )
                    AS TXNAME ( MARKER, CN1,  . . .  , CNK  )
         ) AS TEMPTABLE ( MARKER, WIFTVAL )
    WHERE MARKER = 1 )
```

6) If <inverse distribution function> is specified, then

   a) Let *NVE* be the value of the <inverse distribution function argument>.

   b) If *NVE* is the null value, then the result is the null value.

   c) If *NVE* is less than 0 (zero) or greater than 1 (one), then an exception condition is raised: *data exception — numeric value out of range.*

d) Let *TXA* be the single-column table that is the result of applying the <value expression> simply contained in the <sort specification> to each row of *T1* and eliminating null values. If one or more null values are eliminated, then a completion condition is raised: *warning — null value eliminated in set function*. *TXA* is ordered by the <sort specification> as specified in the General Rules of Subclause 8.2, "<sort specification list>".

e) Let *TXANAME* be an implementation-dependent name for *TXA*.

f) Let *TXCOLNAME* be an implementation-dependent column name for the column of *TXA*.

g) Let *WSP* be obtained from the <sort specification> by replacing the <sort key> with *TXCOLNAME*.

h) Case:

i) If PERCENTILE_CONT is specified, then:

1) Let *ROW0* be the greatest exact numeric value with scale 0 (zero) that is less than or equal to *NVE**(*N*−1). Let *ROWLIT0* be a <literal> representing *ROW0*.

2) Let *ROW1* be the least exact numeric value with scale 0 (zero) that is greater than or equal to *NVE**(*N*−1). Let *ROWLIT1* be a <literal> representing *ROW1*.

3) Let *FACTOR* be a <approximate numeric literal> representing *NVE**(*N*−1)−*ROW0*.

4) The result is the result of the <scalar subquery>

```
( WITH TEMPTABLE(X, Y) AS
        ( SELECT ROW_NUMBER()
                   OVER (ORDER BY WSP) - 1,
                 TXCOLNAME
          FROM TXANAME )
SELECT CAST ( T0.Y + FACTOR * (T1.Y - T0.Y) AS DT )
FROM TEMPTABLE T0, TEMPTABLE T1
WHERE T0.ROWNUMBER = ROWLIT0
  AND T1.ROWNUMBER = ROWLIT1 )
```

NOTE 26 – Although ROW_NUMBER is nondeterministic, the values of T0.Y and T1.Y are determined by this expression. Note that the only column of *TXA* is completely ordered by *WSP*. If *NVE**(*N*−1) is a whole number, then the rows selected from T0 and T1 are the same and the result is just T0.Y. Otherwise, the subquery performs a linear interpolation between the two consecutive values whose row numbers in the ordered set, seen as proportions of the whole, bound the argument of the PERCENTILE_CONT operator.

ii) If PERCENTILE_DISC is specified, then

1) If the <ordering specification> simply contained in *WSP* is DESC, then let *MAXORMIN* be MAX; otherwise let *MAXORMIN* be MIN.

2) Let *NVELIT* be a <literal> representing the value of *NVE*.

3) The result is the result of the <scalar subquery>

8.1 <aggregate function>

```
( SELECT MAXORMIN (TXCOLNAME)
  FROM ( SELECT TXCOLNAME,
                CUME_DIST() OVER (ORDER BY WSP)
  FROM TXANAME ) AS TEMPTABLE (TXCOLNAME, CUMEDIST)
  WHERE CUMEDIST >= NVELIT )
```

Conformance Rules

1) Without Feature T031, "BOOLEAN data type", conforming SQL language shall not contain a <set function type> that specifies EVERY, ANY, or SOME.

2) Without Feature F561, "Full value expressions", or Feature F801, "Full set function", if a <general set function> specifies DISTINCT, then the <value expression> shall be a column reference.

3) Without Feature F441, "Extended set function support", if a <general set function> specifies or implies ALL, then COUNT shall not be specified.

4) Without Feature F441, "Extended set function support", if a <general set function> specifies or implies ALL, then the <value expression> shall contain a column reference that references a column of *T*.

5) Without Feature F441, "Extended set function support", if a <binary set function> is specified, then either the <dependent variable expression> or the <independent variable expression> shall contain a column reference that references a column of *T*.

6) Without Feature F441, "Extended set function support", if the <value expression> simply contained in a <general set function> contains a column reference that is an outer reference, then the <value expression> shall be a column reference.

7) Without Feature F441, "Extended set function support", if the <numeric value expression> simply contained in <dependent variable expression> or <independent variable expression> contains a column reference that is an outer reference, then the <numeric value expression> shall be a column reference.

8) Without Feature F441, "Extended set function support", no column reference contained in an <aggregate function> shall reference a column derived from a <value expression> that generally contains an <aggregate function> *SFS2* without an intervening <routine invocation>.

9) Without Feature S024, "Enhanced structured types", in a <general set function>, if MAX or MIN is specified, then the <value expression> shall not be an ST-ordered declared type.

10) Without Feature S024, "Enhanced structured types", the declared type of a <general set function> shall not be structured type.

11) Without Feature T621, "Enhanced numeric functions", conforming SQL language shall not specify STDDEV_POP, STDDEV_SAMP, VAR_POP, VAR_SAMP, or <binary set function type>.

12) Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify <hypothetical set function> or <inverse distribution function>.

13) Without Feature T612, "Advanced OLAP operators", conforming SQL language shall not specify <filter clause>.

14) Without Feature S024, "Enhanced structured types", if a <set quantifier> of DISTINCT is specified, then the <value expression> shall not be of an ST-ordered declared type.

8.2 <sort specification list>

Function

Specify a sort order.

Format

```
<sort specification list> ::=
       <sort specification> [ { <comma> <sort specification> }... ]

<sort specification> ::=
       <sort key>
           [ <ordering specification> ] [ <null ordering> ]

<sort key> ::=
       <value expression>

<ordering specification> ::=
         ASC
       | DESC

<null ordering> ::=
         NULLS FIRST
       | NULLS LAST
```

Syntax Rules

1) Let *DT* be the declared type of the <value expression> immediately contained in the <sort key> contained in a <sort specification>. *DT* shall not be LOB-ordered, array-ordered, UDT-EC-ordered, or UDT-NC-ordered.

2) If <null ordering> is not specified, then an implementation-defined <null ordering> is implicit. The implementation-defined default for <null ordering> shall not depend on the context outside of <sort specification list>.

Access Rules

None.

General Rules

1) A <sort specification list> defines an ordering of rows, as follows:

   a) Let *N* be the number of <sort specification>s.

   b) Let K_i, 1 (one) $\leq i \leq N$, be the <sort key> contained in the *i*-th <sort specification>.

   c) Each <sort specification> specifies the *sort direction* for the corresponding sort key K_i. If DESC is not specified in the *i*-th <sort specification>, then the sort direction for K_i is ascending and the applicable <comp op> is the <less than operator>. Otherwise, the sort direction for K_i is descending and the applicable <comp op> is the <greater than operator>.

8.2 <sort specification list>

d) Let P be any row of the multiset of rows to be ordered, and let Q be any other row of the same multiset of rows.

e) Let PV_i and QV_i be the values of K_i in P and Q, respectively. The relative position of rows P and Q in the result is determined by comparing PV_i and QV_i according to the rules of Subclause 8.2, "<comparison predicate>", in ISO/IEC 9075-2, where the <comp op> is the applicable <comp op> for K_i, with the following special treatment of null values.

Case:

i) If PV_i and QV_i are both null, then they are considered equal to each other.

ii) If PV_i is null and QV_i is not null, then

Case:

1) If NULLS FIRST is specified or implied, then PV_i <comp op> QV_i is considered to be *true* .

2) If NULLS LAST is specified or implied, then PV_i <comp op> QV_i is considered to be *false* .

iii) If PV_i is not null and QV_i is null, then

Case:

1) If NULLS FIRST is specified or implied, then PV_i <comp op> QV_i is considered to be *false* .

2) If NULLS LAST is specified or implied, then PV_i <comp op> QV_i is considered to be *true* .

f) PV_i is said to *precede* QV_i if the value of the <comparison predicate> "PV_i <comp op> QV_i" is *true* for the applicable <comp op>.

g) If PV_i and QV_i are not null and the result of "PV_i <comp op> QV_i" is *unknown* , then the relative ordering of PV_i and QV_i is implementation-dependent.

h) The relative position of row P is before row Q if PV_n precedes QV_n for some n, 1 (one) $\leq n \leq N$, and PV_i is not distinct from QV_i for all $i < n$.

i) Two rows that are not distinct with respect to the <sort specification>s are said to be *peers* of each other. The relative ordering of peers is implementation-dependent.

Conformance Rules

1) Without Feature T611, "Elementary OLAP operators", conforming SQL language shall not specify <null ordering>.

2) Without Feature S024, "Enhanced structured types", the declared type of the <value expression> immediately contained in the <sort key> shall not be ST-ordered.

9 Schema definition and manipulation

9.1 <drop table constraint definition>

Function

Destroy a constraint on a table.

Format

```
!! No additional Format items.
```

Syntax Rules

1) Replace SR 4) If *V* is a view that contains a <query specification> *QS* that contains a column reference to a column *C* in its <select list> that is not contained in an aggregated argument of a <set function specification>, and if *G* is the set of columns defined by the <grouping column reference list> of *QS*, and if the table constraint *TC* is needed to conclude that $G \mapsto C$ is a known functional dependency in *QS*, then *V* is said to be *dependent on TC*.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

9.2 <drop user-defined ordering statement>

Function

Destroy a user-defined ordering method.

Format

```
!! No additional Format items
```

Syntax Rules

1) Insert this subrule after SR 4)j) A <sort key> that simply contains a <value expression> whose declared type is *S*-ordered.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

10 SQL-client modules

10.1 Calls to an <externally-invoked procedure>

Function

Define the call to an <externally-invoked procedure> by an SQL-agent.

Syntax Rules

1) Insert into SR 2)e)

```
DATA_EXCEPTION_INVALID_ARGUMENT_FOR_NATURAL_LOGARITHM:
        constant SQLSTATE_TYPE :="2201E";
DATA_EXCEPTION_INVALID_ARGUMENT_FOR_POWER_FUNCTION:
        constant SQLSTATE_TYPE :="2201F";
DATA_EXCEPTION_INVALID_ARGUMENT_FOR_WIDTH_BUCKET_FUNCTION:
        constant SQLSTATE_TYPE :="2201G";
DATA_EXCEPTION_INVALID_PRECEDING_OR_FOLLOWING_SIZE_IN_WINDOW_FUNCTION:
        constant SQLSTATE_TYPE :="22013";
```

(Blank page)

11 Data manipulation

11.1 <declare cursor>

Function

Define a cursor

Format

```
!! No additional Format items
```

Delete the definition of <sort specification list>

Delete the definition of <sort specification>

Delete the definition of <sort key>

Delete the definition of <ordering specification>

NOTE 27 – The deleted definitions are now placed in Subclause 8.2, "<sort specification list>".

Syntax Rules

1) Replace SR 18)f)i)2)A)II) *QS* shall not immediately contain the <set quantifier> DISTINCT or directly contain one or more <set function specification>s.

2) Delete SR 19)

3) Delete SR 22)

NOTE 28 – The deleted Syntax Rules are now found in Subclause 8.2, "<sort specification list>".

Access Rules

No additional Access Rules

General Rules

1) Replace GR 2) If an <order by clause> is specified, then the ordering of rows of the result is determined by the <sort specification list>. The result table specified by the <cursor specification> is *TS* with all extended sort key columns (if any) removed.

Conformance Rules

No additional Conformance Rules.

11.2 <select statement: single row>

Function

Retrieve values from a specified row of a table.

Format

!! *No additional Format items*

Syntax Rules

1) Delete SR 5)

2) Insert after SR 6) The <select statement: single row> is possibly nondeterministic if *S* is possibly nondeterministic.

Access Rules

No additional Access Rules

General Rules

No additional General Rules

Conformance Rules

No additional Conformance Rules.

12 Dynamic SQL

12.1 <prepare statement>

Function

Retrieve values from a specified row of a table.

Format

```
!! No additional Format items
```

Syntax Rules

No additional Syntax Rules

Access Rules

No additional Access Rules

General Rules

1) Insert after GR 6)a)xxvii) If *DP* is contained in a <window frame preceding> or a <window frame following> contained in a <window specification> *WS*, then

   Case:

   1) If *WS* specifies ROWS, then *DT* is NUMERIC (*MP*, 0).

   2) Otherwise, let *SDT* be the data type of the single <sort key> contained in *WS*.

      Case:

      A) If *SDT* is a numeric type, then *DT* is *SDT*.

      B) If *SDT* is DATE, then *DT* is INTERVAL DAY.

      C) If *SDT* is TIME(*P*) WITHOUT TIME ZONE or TIME(*P*) WITH TIME ZONE, then *DT* is INTERVAL HOUR TO SECOND(*P*).

      D) If *SDT* is TIMESTAMP(*P*) WITHOUT TIME ZONE or TIMESTAMP(*P*) WITH TIME ZONE, then DT is INTERVAL DAY TO SECOND(*P*).

      E) If *SDT* is an interval type, then *DT* is *SDT*.

Conformance Rules

No additional Conformance Rules.

(Blank page)

13 Information Schema

13.1 Definition of SQL built-in functions

Function

Define the SQL built-in functions.

Definition

Add the following Definitions

```
CREATE FUNCTION "CEIL" (
    N       numeric_type )
    RETURNS NUMERIC ( MP , 0 )
    SPECIFIC CEIL
    RETURN CEIL ( N ) ;

CREATE FUNCTION "CEILING" (
    N       numeric_type )
    RETURNS NUMERIC ( MP, 0 )
    SPECIFIC CEILING
    RETURN CEILING ( N ) ;

CREATE FUNCTION "EXP" (
    N       numeric_type )
    RETURNS DOUBLE PRECISION
    SPECIFIC EXP
    RETURN EXP ( N ) ;

CREATE FUNCTION "FLOOR" (
    N       numeric_type )
    RETURNS NUMERIC ( MP, 0)
    SPECIFIC FLOOR
    RETURN FLOOR ( N ) ;

CREATE FUNCTION "LN" (
    N       numeric_type )
    RETURNS DOUBLE PRECISION
    SPECIFIC LN
    RETURN LN ( N ) ;

CREATE FUNCTION "POWER" (
    M       numeric_type,
    N       numeric_type)
    RETURNS DOUBLE PRECISION
    SPECIFIC POWER
    RETURN POWER ( M, N ) ;

CREATE FUNCTION "SQRT" (
    N       numeric_type )
    RETURNS DOUBLE PRECISION
    SPECIFIC SQRT
    RETURN SQRT ( N ) ;
```

13.1 Definition of SQL built-in functions

```
CREATE FUNCTION "WIDTH_BUCKET" (
    A       numeric_type,
    B       numeric_type,
    C       numeric_type,
    D       numeric_type )
    RETURNS NUMERIC (MP, 0)
    SPECIFIC WIDTH_BUCKET
    RETURN WIDTH_BUCKET ( A, B, C, D) ;

GRANT EXECUTE ON SPECIFIC FUNCTION INFORMATION_SCHEMA."CEIL"
TO PUBLIC;

GRANT EXECUTE ON SPECIFIC FUNCTION INFORMATION_SCHEMA."CEILING"
TO PUBLIC;

GRANT EXECUTE ON SPECIFIC FUNCTION INFORMATION_SCHEMA."EXP"
TO PUBLIC;

GRANT EXECUTE ON SPECIFIC FUNCTION INFORMATION_SCHEMA."FLOOR"
TO PUBLIC;

GRANT EXECUTE ON SPECIFIC FUNCTION INFORMATION_SCHEMA."LN"
TO PUBLIC;

GRANT EXECUTE ON SPECIFIC FUNCTION INFORMATION_SCHEMA."POWER"
TO PUBLIC;

GRANT EXECUTE ON SPECIFIC FUNCTION INFORMATION_SCHEMA."SQRT"
TO PUBLIC;

GRANT EXECUTE ON SPECIFIC FUNCTION INFORMATION_SCHEMA."WIDTH_BUCKET"
TO PUBLIC;
```

Description

1) Insert this Description *numeric_type* is the implementation-defined numeric type that has the highest type precedence among all numeric types supported by that implementation.

14 Status codes

14.1 SQLSTATE

Table 2—SQLSTATE class and subclass values

Category	Condition	Class	Subcondition	Subclass
	All alternatives from ISO/IEC 9075-2			
	All alternatives from ISO/IEC 9075-5			
X	data exception	22	invalid argument for natural logarithm	01E
			invalid argument for power function	01F
			invalid argument for width bucket function	01G
			invalid preceding or following size in window function	013

(Blank page)

