

Final Committee Draft ISO/IEC FPDAM 9075 Amd 1	
Date: 2000-05-01	Reference number: ISO/JTC 1/SC 32 N 0379
Supersedes document SC 32 N 0292	

THIS DOCUMENT IS STILL UNDER STUDY AND SUBJECT TO CHANGE. IT SHOULD NOT BE USED FOR REFERENCE PURPOSES.

ISO/IEC JTC 1/SC 32 Data Management and Interchange Secretariat: USA (ANSI)	Circulated to P- and O-members, and to technical committees and organizations in liaison for voting (P-members only) by: 2000-05-01 Please return all votes and comments in electronic form directly to the SC 32 Secretariat by the due date indicated.
--	---

ISO/IEC FPDAM 9075 : Amd 1 Title: Information technology- Database languages SQL - Amendment 1: SQL/OLAP (for SQL:1999) Project: 1.03.04.01.01.00

Introductory note: The attached document is hereby submitted for a four-month letter ballot to the National Bodies of ISO/IEC JTC 1/SC 32. The ballot starts 2000-01-01; ITTF must have the document two weeks before the balloting can start. Medium: E No. of pages: 97

Address Reply to: Douglas Mann, Secretariat, ISO/IEC JTC 1/SC 32, Pacific Northwest National Laboratory, 901 D Street, SW., Suite 900, Washington, DC, 20024-2115, United States of America
Telephone: +1 703 575 2114; Facsimile: +1 703 681 9180; E-mail: MannD@battelle.org

SC32 N00379

**Final Proposed Draft Amendment — Database Language SQL —
Amendment 1: On-Line Analytical Processing (SQL/OLAP)
«Amendment 1»**

November 1999

For FPDAM Ballot

Contents	Page
Foreword	vii
Introduction	ix
1 Scope	1
2 Normative references	3
3 Definitions, notations, and conventions	5
3.1 Definitions	5
3.1.1 Definitions provided in Amendment 1	5
3.2 Notations	5
3.3 Conventions	5
3.3.1 Relationships to other parts of ISO/IEC 9075	5
3.3.1.1 Clause, Subclause, and Table relationships	5
4 Concepts	9
4.1 Numbers	9
4.1.1 Operations involving numbers	9
4.2 Data analysis operations (involving tables)	9
4.2.1 Grouped table functions	9
4.2.2 Windowed table functions	10
4.2.3 Aggregate functions	11
4.3 Tables	13
4.3.1 Windowed tables	13
4.4 SQL-invoked routines	14
5 Lexical elements	15
5.1 <token> and <separator>	15
5.2 Names and identifiers	17
6 Scalar expressions	19
6.1 <numeric value function>	19
6.2 <set function specification>	23
6.3 <windowed table function>	28
6.4 <value expression>	32

7	Query expressions	33
7.1	<table expression>	33
7.2	<joined table>	34
7.3	<where clause>	35
7.4	<having clause>	36
7.5	<window clause>	38
7.6	<query specification>	48
8	Additional common elements	51
8.1	<aggregate function>	51
8.2	<sort specification list>	58
9	Schema definition and manipulation	61
9.1	<drop routine statement>	61
9.2	<drop user-defined ordering statement>	63
10	SQL-client modules	65
10.1	Calls to an <externally-invoked procedure>	65
11	Data manipulation	67
11.1	<declare cursor>	67
11.2	<select statement: single row>	68
12	Dynamic SQL	69
12.1	<prepare statement>	69
13	Information Schema	71
13.1	Definition of SQL built-in functions	71
14	Status codes	73
14.1	SQLSTATE	73
15	Conformance	75
15.1	General conformance requirements	75
Annex A	SQL conformance summary	77
Annex B	Implementation-defined elements	79
Annex C	Implementation-dependent elements	81
Annex D	SQL feature and package taxonomy	83
Index		Index1

TABLES

Tables	Page
1 Clause, Subclause, and Table relationships	5
2 SQLSTATE class and subclass values	73
3 Implied feature relationships	75
4 SQL/OLAP feature taxonomy for features outside Core SQL	83

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.

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.

International Standard ISO/IEC 9075, was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 32, *Data management and interchange*.

ISO/IEC 9075 consists of the following parts, under the general title *Information technology — Database languages — SQL*:

- *Part 1: Framework (SQL/Framework)*
- *Part 2: Foundation (SQL/Foundation)*
- *Part 3: Call-Level Interface (SQL/CLI)*
- *Part 4: Persistent Stored Modules (SQL/PSM)*
- *Part 5: Host Language Bindings (SQL/Bindings)*

Annexes A, B, C, and D of this amendment to ISO/IEC 9075 are for information only.

Introduction

The organization of this amendment to ISO/IEC 9075 is as follows:

- 1) Clause 1, “Scope”, specifies the scope of this amendment to ISO/IEC 9075.
- 2) Clause 2, “Normative references”, identifies additional standards that, through reference in this amendment to ISO/IEC 9075, constitute provisions of this amendment to ISO/IEC 9075.
- 3) Clause 3, “Definitions, notations, and conventions”, defines the notations and conventions used in this amendment to ISO/IEC 9075.
- 4) Clause 4, “Concepts”, presents concepts used in the definition of OLAP facilities.
- 5) Clause 5, “Lexical elements”, defines a number of lexical elements used in the definition of OLAP facilities.
- 6) Clause 6, “Scalar expressions”, defines a number of scalar expressions used in the definition of OLAP facilities.
- 7) Clause 7, “Query expressions”, defines the elements of the language that produce rows and tables of data as used in OLAP facilities.
- 8) Clause 8, “Additional common elements”, defines additional common elements used in the definition of OLAP facilities.
- 9) Clause 9, “Schema definition and manipulation”, defines the schema definition and manipulation statements associated with the definition of OLAP 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 OLAP 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 OLAP facilities.
- 14) Clause 14, “Status codes”, defines SQLSTATE values related to OLAP facilities.
- 15) Clause 15, “Conformance”, defines the criteria for conformance to this amendment to ISO/IEC 9075.
- 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 to ISO/IEC 9075 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.

SC32 N00379 — FPDAM 9075:1999/AM1

- 18) Annex C, “Implementation-dependent elements”, is an informative Annex. It lists those features for which the body of this amendment to ISO/IEC 9075 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 to ISO/IEC 9075 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.

In the text of this amendment to ISO/IEC 9075, 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 —

Amendment 1:

On-Line Analytical Processing (SQL/OLAP)

1 Scope

This amendment to International Standard ISO/IEC 9075 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:

— To Be Supplied

NOTE 1 – The context for this amendment to ISO/IEC 9075 is described by the Reference Model of Data Management (ISO/IEC 10032:1993).

2 Normative references

The following standards contain provisions that, through reference in this text, constitute provisions of this amendment to ISO/IEC 9075. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this amendment to ISO/IEC 9075 are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Members of IEC and ISO 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 9075-3:1999, *Information technology — Database languages — SQL — Part 3: Call-Level Interface (SQL/CLI)*.

ISO/IEC 9075-4:1999, *Information technology — Database languages — SQL — Part 4: Persistent Stored Modules (SQL/PSM)*.

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

3 Definitions, notations, and conventions

3.1 Definitions

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

3.1.1 Definitions provided in Amendment 1

To Be Supplied

3.2 Notations

Insert this paragraph The syntax notation used in this amendment to ISO/IEC 9075 is an extended version of BNF ("Backus Normal Form" or "Backus Naur Form"). This version of BNF is fully described in Subclause 6.1, "Notation", of ISO/IEC 9075-1.

3.3 Conventions

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

3.3.1 Relationships to other parts of ISO/IEC 9075

3.3.1.1 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 3.1.1, "Definitions provided in Amendment 1"	<i>(none)</i>	<i>(none)</i>

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 3.2, "Notations"	Subclause 3.3, "Conventions"	ISO/IEC 9075-2
Subclause 3.3, "Conventions"	Subclause 3.3, "Conventions"	ISO/IEC 9075-2
Subclause 3.3.1, "Relationships to other parts of ISO/IEC 9075"	<i>(none)</i>	<i>(none)</i>
Subclause 3.3.1.1, "Clause, Subclause, and Table relationships"	<i>(none)</i>	<i>(none)</i>
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, "Data analysis operations (involving tables)"	<i>(none)</i>	<i>(none)</i>
Subclause 4.2.1, "Grouped table functions"	<i>(none)</i>	<i>(none)</i>
Subclause 4.2.2, "Windowed table functions"	<i>(none)</i>	<i>(none)</i>
Subclause 4.2.3, "Aggregate functions"	<i>(none)</i>	<i>(none)</i>
Subclause 4.3, "Tables"	Subclause 4.16, "Tables"	ISO/IEC 9075-2
Subclause 4.3.1, "Windowed tables"	<i>(none)</i>	<i>(none)</i>
Subclause 4.4, "SQL-invoked routines"	Subclause 4.23, "SQL-invoked routines"	ISO/IEC 9075-2
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, "<numeric value function>"	Subclause 6.17, "<numeric value function>"	ISO/IEC 9075-2
Subclause 6.2, "<set function specification>"	Subclause 6.16, "<set function specification>"	ISO/IEC 9075-2
Subclause 6.3, "<windowed table function>"	<i>(none)</i>	<i>(none)</i>
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

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 7.1, "<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
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 routine statement>"	Subclause 11.51, "<drop routine statement>"	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"	Subclause 4.21, "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 4.24, "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

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 15.1, "General conformance requirements"	Subclause 23.1, "General conformance requirements"	ISO/IEC 9075-2
Annex A, "SQL conformance summary"	Appendix A, "SQL Conformance Summary"	ISO/IEC 9075-2
Annex B, "Implementation-defined elements"	Appendix B, "Implementation-defined elements"	ISO/IEC 9075-2
Annex C, "Implementation-dependent elements"	Appendix C, "Implementation-dependent elements"	ISO/IEC 9075-2
Annex D, "SQL feature and package taxonomy"	Appendix F, "SQL feature and package taxonomy"	ISO/IEC 9075-2
Table 1, "Clause, Subclause, and Table relationships"	<i>(none)</i>	<i>(none)</i>
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

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 1 and final argument, by assigning the first argument to an equi-width bucket partitioning the range of numbers between the second and third arguments.

4.2 Data analysis operations (involving tables)

A data analysis function is a function that generally 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 grouped table 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 windowed table function, which is invoked on a windowed table and computes a rank, row number or windowed table aggregate function.

4.2.1 Grouped table functions

A grouped table 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 grouped table function is one of:

- The *grouping operation*.

4.2 Data analysis operations (involving tables)

— A *grouped table aggregate function*.

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.

4.2.2 Windowed table functions

A windowed table 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. Windowed table 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 windowed table function is one of:

- A rank function.
- A distribution function.
- The row number function.
- A windowed table 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 2 – This implies that if two or more rows are not distinct with respect to the windows 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 3 – 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 partition of *R* defined by a window 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 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 windowed table aggregate functions compute an aggregate value (COUNT, SUM, AVG, *etc.*), the same as a grouped table aggregate function, except that the computation aggregates over the window frame of a row rather than over a group of a grouped table.

4.2.3 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 grouped table 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.

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

Every other grouped table aggregate function may be classified as a *unary grouped table aggregate function*, a *binary grouped table aggregate functions*, an *inverse distribution*, or a *what-if function*. Every unary grouped aggregate function takes an arbitrary <value expression> as the argument; most unary grouped aggregate functions can optionally 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 .
- 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.

SC32 N00379 — FPDAM 9075:1999/AM1

4.2 Data analysis operations (involving tables)

- 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 grouped table 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 grouped table 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 take 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, and the 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, and interpolating the value of the value expression evaluated for these rows.
- If PERCENTILE_DISC is specified, by treating the group as a partition of the CUME_DIST OLAP function, using the specified ordering of the value expression, and returning the first value expression whose cumulative distribution value is greater than or equal to the argument.

The what-if functions are related to the OLAP functions RANK, DENSE_RANK, PERCENT_RANK and CUME_DIST, and using 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 partition of the corresponding OLAP function. The result of the what-if function is the value of the eponymous OLAP function for the hypothetical “row” that contributes *A* to the collection.

4.3 Tables

4.3.1 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 framings, which are multisets of rows used in the definition of <window table 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 partition clause>. Window partitioning is optional; if omitted, there is a single window partition consisting of all the rows in the result.

NOTE 4 – If a <table expression> is grouped and also has a window, there is a syntactic transformation that segregates the grouping into a <derived table>, so that the window partitions consists of rows of the <derived table> rather than groups of rows.

A window may define an 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.

4.3 Tables

- A starting row, which may be the first row of the window partition, 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, 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*, consisting of:

- 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—that is, a <window partition clause> (if not specified, a zero-length string).
- The window ordering—that is, <window order clause> (if not specified, a zero-length string).
- The window framing— that is, a <window frame clause> (if not specified, a zero-length string).

Two <windowed table function>s are computed using the same (possibly non-deterministic) window ordering of the rows if any of the following are true:

- The <windowed table function>s identify the same window structure descriptor.
- The <windowed table function>s' window structure descriptors have window partitionings 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 orderings 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.
- The window structure descriptor of one <windowed table function> is the ordering window of the other <windowed table function>, or both window structure descriptors identify the same ordering window.

4.4 SQL-invoked routines

Insert after the sixth paragraph A <windowed table function> *OF* is said to be *dependent* on an SQL-invoked routine *SR* if and only if all the following are true:

- *SR* is the ordering function included in the user-defined descriptor of a user-defined type *UDT*.
- Either there is a column reference whose declared type is *UDT* simply contained in the window partitioning of the window structure descriptor referenced by *OF*, or there is a <value expression> whose declared type is *UDT* simply contained in the window ordering of the window structure descriptor referenced by *OF*.

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

    | 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

    | STDDEV_POP | STDDEV_SAMP | SQRT

    | TIES

    | UNBOUNDED

| VAR_POP | VAR_SAMP

| WIDTH_BUCKET

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

```

SC32 N00379 — FPDAM 9075:1999/AM1
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 functions”, conforming SQL language shall not specify <window name>.

6 Scalar expressions

6.1 <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> ::=

```

SC32 N00379 — FPDAM 9075:1999/AM1

6.1 <numeric value function

<numeric value expression>

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

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

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, 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, 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 more 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 — numeric value out of range*.
 - 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 funtion> 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 — numeric value out of range*.

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 — numeric value out of range*.

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

$$\text{EXP}(NVEE * \text{LN}(-NVEB))$$

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

$$-\text{EXP}(NVEE * \text{LN}(-NVEB))$$

h) Otherwise, the result is the result of

$$\text{EXP}(NVEE * \text{LN}(NVEB))$$

4) Insert this GR If <floor function> is specified, then the result is the greatest exact numeric value with scale 0 (zero) that is less than or equal to the value of the <numeric value expression>. 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 the result is the least exact numeric value with scale 0 (zero) that is greater than or equal to the value of the <numeric value expression>. 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.

b) If WBC is less than or equal to 0 (zero), then an exception condition is raised: *data exception — numeric value out of range*.

c) If $WBB1$ equals $WBB2$, then an exception condition is raised: *data exception — numeric value out of range*.

SC32 N00379 — FPDAM 9075:1999/AM1

6.1 <numeric value function

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

Case:

- i) If WBO is less than $WBB1$, then the result is 1 (one).
- ii) If WBO is greater than or equal to $WBB2$, then the result is WBC .
- 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 1 (one).
- ii) If WBO is less than or equal to $WBB2$, then the result is WBC .
- 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 functions”, conforming SQL language shall not specify <width bucket function>.

6.2 <set function specification>

Function

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

Format

```
<set function specification> ::=
    <unordered set function>
    | <ordered set function>

<unordered set function> ::=
    <aggregate function>
    | <grouping operation>

<unordered set function> ::= <aggregate function>
    | <grouping operation>

<ordered set function> ::=
    <what-if set function>
    | <inverse distribution function>

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

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

<what-if 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
```

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>”.

6.2 <set function specification>

Syntax Rules

- 1) Delete SR 1) through 3)
- 2) 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>.
- 3) 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>.
- 4) Insert this SR If <what-if set function> is specified, then:
 - a) The <what-if set function> shall not contain a <windowed table function>, a <set function specification>, or a <subquery>.
 - b) If a <value expression> simply contained in the <what-if 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 <what-if 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 <what-if 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) 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.
- 5) Insert this SR If <set function specification> specifies <inverse distribution function>, then:
 - a) The <within group specification> shall contain a single <sort specification>.
 - b) The <inverse distribution function> shall not contain an <windowed table 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) If PERCENTILE_CONT is specified, then the declared type of the <sort key> simply contained in the <sort specification> shall be numeric or interval.
 - e) The declared type of the result is the declared type of the <value expression> simply contained in the <sort specification>.
- 6) Insert after SR5)a) A <window 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 SR16)

Access Rules

No additional Access Rules.

General Rules

- 1) Delete GR1) and GR2)
- 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>.
- 4) Insert this GR If <what-if function type> is specified, then
- a) Let *WIFT* be the <rank function type>.
 - b) The argument source *T* is a table or group of a grouped table as specified in Subclause 7.10, "<having clause>", and Subclause 7.6, "<query specification>". Let *TNAME* be an implementation-dependent name for *T*.
 - c) Let *K* be the number of <value expression>s simply contained in <what-if value expression list>.
 - d) Let VE_1, \dots, VE_K be the <value expression>s simply contained in the <what-if value expression list>.
 - e) Let *MARKER* and CN_1, \dots, CN_K be distinct implementation-dependent column names.
 - f) Let SP_1, \dots, 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>
- a)

SC32 N00379 — FPDAM 9075:1999/AM1

6.2 <set function specification>

```
( 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 )
```

- 5) Insert this GR 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) The argument source *T* is a table or group of a grouped table as specified in Subclause 7.4, “<having clause>”, and Subclause 7.6, “<query specification>”.
 - e) 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 *T*. *TXA* is ordered by the <sort specification> as specified in the General Rules of Subclause 8.2, “<sort specification list>”.
 - f) Let *TXANAME* be an implementation-dependent name for *TXA*.
 - g) Let *TXCOLNAME* be an implementation-dependent column name for the column of *TXA*.
 - h) Let *WSP* be obtained from the <sort specification> by replacing the <sort key> with *TXCOLNAME*.
 - i) 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 literal representing $NVE*(N-1)-ROW0$.
 - 4) Let *DT* be the declared type of the column of *TXA*.
 - 5) The result is the result of the <scalar subquery>

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

SC32 N00379 — FPDAM 9075:1999/AM1
6.2 <set function specification>

NOTE 6 – 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 WPS. 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>

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

Conformance Rules

- 1) Delete CRs 1) through 6)
- 2) Delete CRs 8) and 9)
- 3) Insert this CR Without Feature T612, “Advanced OLAP functions”, conforming SQL language shall not specify <what-if set function> or <inverse distribution function>.

6.3 <windowed table function>**6.3 <windowed table function>****Function**

Specify an OLAP function.

Format

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

```
<windowed table 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) Let *OF* be the <windowed table function>.
- 2) Case:
 - a) If *OF* is contained 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*.
- 3) *OF* shall not contain an outer reference or a <subquery>.
- 4) Let *WNS* be the <window name or specification>. Let *WDX* be a window structure descriptor that describes the window defined by *WNS*.
- 5) If <rank function type> or ROW_NUMBER is specified, then:
 - a) The window ordering *WOC* of *WDX* shall not be a zero-length string.
 - b) The window framing of *WDX* shall be a zero-length string.

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 WNS1` 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, \dots, VE_N be an enumeration of the <value expression>s that are <sort key>s simply contained in *WOC*.
- ii) `DENSE_RANK() OVER WNS1` is equivalent to the <windowed table function>:

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

f) `ROW_NUMBER() OVER WNS` is equivalent to the <windowed table function>:

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

g) Let *ANT1* be an approximate numeric type with implementation-defined precision. `PERCENT_RANK() OVER WNS1` 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 WNS1` is equivalent to:

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

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

NOTE 7 – 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.

7) 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.

SC32 N00379 — FPDAM 9075:1999/AM1

6.3 <windowed table function>

8) 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 <windowed table 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*

- 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*

9) If the window ordering or the window framing of the window descriptor that describes the <window name or specification> is not a zero-length string, then no <aggregate function> simply contained in <windowed table function> shall specify DISTINCT.

Access Rules

None.

General Rules

- a) The value of <windowed table function> is the value of the <aggregate function>.

Conformance Rules

- 1) Without Feature T611, “Elementary OLAP functions”, conforming SQL language shall not specify a <windowed table function>.
- 2) Without Feature T612, “Advanced OLAP functions”, conforming SQL language shall not specify <window name>.
- 3) Without Feature T612, “Advanced OLAP functions”, conforming SQL language shall not specify PERCENT_RANK or CUME_DIST.

6.4 <value expression>

6.4 <value expression>

Function

Specify a value.

Format

```
<nonparenthesized value expression primary> ::=  
    !! All alternatives from ISO/IEC 9075-2  
    | <windowed table 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 <windowed table function>, or the effective returns type of the immediately contained <routine invocation>, respectively.

Access Rules

None.

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 <windowed table function>.

Conformance Rules

- 1) Insert this CR Without Feature T611, “Elementary OLAP functions”, a <value expression> shall not be a <windowed table function>.

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 functions”, conforming SQL language shall not specify a <window clause>.

7.2 <joined table>

Function

Reference a table.

Format

!! No additional Format items

Syntax Rules

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

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 3) Each column reference directly contained in the <search condition> shall be one of the following:
 - a) An unambiguous reference to a column that is functionally dependent on G .
 - b) An outer reference.
 - c) Contained in a <parameterized set function>.
 - d) Contained in the <within group specification> of an <ordered set function>.
- 2) Replace SR 4) Each column reference contained in a <subquery> in the <search condition> that references a column of T shall be one of the following:
 - a) A reference to a column that is functionally dependent on G .
 - b) Contained in a <parameterized set function>.
 - c) Contained in the <within group specification> of an <ordered set function>.
- 3) Insert after SR 4) The <search condition> shall not contain a <windowed table function> without an intervening <subquery>.

Access Rules

No additional Access Rules

General Rules

No additional General Rules

Conformance Rules

- 1) Replace CR 1) Without Feature T301, “Functional dependencies”, each column reference directly contained in the <search condition> shall be one of the following:
 - a) An unambiguous reference to a grouping column of T .
 - b) An outer reference.

- c) Contained in a <parameterized set function>.
 - d) Contained in the <within group specification> of an <ordered set function>.
- 2) 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 a <parameterized set function>.
 - c) Contained in the <within group specification> of an <ordered set function>.

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>
```

```

| 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 8 – 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) 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 be one of the following:
 - a) A reference to a column that is functionally dependent on *G*.
 - b) Contained in a <parameterized set function>.
 - c) Contained in the <within group specification> of an <ordered set function>.
- 6) A <window clause> shall not contain a <windowed table function> without an intervening <subquery>.

SC32 N00379 — FPDAM 9075:1999/AM1

7.5 <window clause>

- 7) 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.
- 8) 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

- 9) 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 that is not a zero-length string, then *WDEF* shall not specify <window order clause>.
 - e) *WDX* shall not have a window framing that is not a zero-length string.
- 10) 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).
- 11) 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>.

- 12) Two window structure descriptors *WD1* and *WD2* are *order-equivalent* if all of the following conditions are met:
- a) Let $WPCR1_i$, $1 \text{ (one)} \leq i \leq N1$, and $WPCR2_i$, $1 \text{ (one)} \leq i \leq N2$, be enumerations of the <window partition column reference>s contained in the window partitioning 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 $SS1_i$, $1 \text{ (one)} \leq i \leq M1$, and $SS2_i$, $1 \text{ (one)} \leq i \leq M2$, be enumerations of the <sort specification>s contained in the window ordering 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* has no <windowed table 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 9 – Although it is permissible to have a <group by clause> or a <having clause> with a <window clause>, if there are any <windowed table 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) Let *EWN* be the <existing window name> simply contained in *WDEF*, if any. Let *WDX* be the window structure descriptor identified by *EWN* if <existing window name> is specified.
 - 3) The ordering window name of *WDESC* is *EWN*, if <existing window name> is specified and if the <window order clause> of *WDX* is not a zero-length string; otherwise, there is no ordering window name.
 - 4) *WDESC*'s window partitioning is the <window partition clause> simply contained in *WDEF*, if one is specified; otherwise it. is the <window partition clause> of *WDX*, if <existing window name> is specified; otherwise, it is a zero-length string.

SC32 N00379 — FPDAM 9075:1999/AM1

7.5 <window clause>

- 5) *WDESC*'s window ordering is the <window order clause> simply contained in *WDEF*, if one is specified; otherwise, it is the <window order clause> of *WDX*, if <existing window name> is specified; otherwise it is a zero-length string.
 - 6) *WDESC*'s window framing is the <window frame clause> simply contained in *WDEF*, if <window frame clause> is specified; otherwise it is a zero-length string.
- 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 the window partitioning is a zero-length string, then the window partition of *R* is the entire result *RTE*.
 - 4) *WD* also defines an 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. If the window ordering is a zero-length string, then the window ordering is entirely implementation-dependent, and all rows are peers. Although the 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. Let *WFC* be the window framing of *WD*. *WF* is defined as follows:
 - a) If *WFC* is a zero-length string, then
Case:
 - i) If the window ordering of *WD* is a zero-length string, then *WF* is the window partition of *R*.
 - ii) Otherwise, *WF* consists of all rows of the partition of *R* that precede *R* or are peers of *R* in the ordering of the partition defined by the window ordering.
 - b) Otherwise, let *WF* initially be the window partition of *R* defined by *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) Case:

NOTE 10 – 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 *VIP* be the value of the <unsigned value specification>.
 - I) If *VIP* is negative or the null value, then an exception condition is raised: *data exception—invalid preceding or following size in OLAP function*.

- II) Otherwise, let *SK* be the only <sort key> contained in the window ordering 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 specifies DESC, then let *BOUND* be the value $VSK + VIP$. 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 $VSK - VIP$. 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 ordering defined by *WD*.
- C) If *WFB1* specifies <window frame following>, then let *VIF* be the value of the <unsigned value specification>.
- I) If *VIF* is negative or the null value, then an exception condition is raised: *data exception—invalid preceding or following size in OLAP function*.
 - II) Otherwise, let *SK* be the only <sort key> contained in the window ordering 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 order clause> specifies DESC, then let *BOUND* be the value $VSK - VIF$. Remove from *WF* all rows *R2* such that the value of *SK* in row *R2* is greater than *BOUND*.

SC32 N00379 — FPDAM 9075:1999/AM1

7.5 <window clause>

- 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*.

2) Case:

NOTE 11 – 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 OLAP function*.

- II) Otherwise, let *SK* be the only <sort key> contained in the <window order 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 $VSK+V2P$. 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 OLAP function*.

- II) Otherwise, let *SK* be the only <sort key> contained in the <window order 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 12 – 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 *VIP* be the value of the <unsigned value specification>.
 - I) If *VIP* is negative or the null value, then an exception condition is raised: *data exception—invalid preceding or following size in OLAP function*.
 - II) Otherwise, remove from *WF* all rows that are more than *VIP* 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 13 – 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 *VIF* be the value of the <unsigned value specification>.
 - I) If *VIF* is negative or the null value, then an exception condition is raised: *data exception—invalid preceding or following size in OLAP function*.
 - II) Otherwise, remove from *WF* all rows that precede the current row and all rows that are less than *VIF* rows following the current row in the window ordering defined by *WD*.

NOTE 14 – If *VIF* is zero, then the current row is not removed from *WF* by this step; otherwise, the current row is removed from *WF*.

SC32 N00379 — FPDAM 9075:1999/AM1

7.5 <window clause>

2) Case:

NOTE 15 – 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 OLAP 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 16 – 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 17 – 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 OLAP 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

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

NOTE 19 – 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 functions”, conforming SQL language shall not specify <window specification>.

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

- 3) Without Feature T612, “Advanced OLAP functions”, 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 one of the following:
 - a) A reference to a grouping column of T .
 - b) Contained in a <parameterized set function>.
 - c) Contained in the <within group specification> of an <ordered set function>.
- 5) Without Feature T612, “Advanced OLAP functions”, conforming SQL language shall not specify <window framing exclusion>.

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 <windowed table 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 <windowed table function>.
- 3) Insert this SR A grouped, windowed query *GWQ* is transformed to an equivalent <query specification> as follows:
 - a) If *GWQ* simply contains a <group by clause> *GBY*, then *GBY* shall not contain ROLLUP, CUBE, or GROUPING SETS.
 - b) If *GWQ* contains an <in-line window specification>, then apply the syntactic transformation specified in Subclause 6.3, “<windowed table function>”.
 - c) 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.
 - d) Let *GWQ2* be the result of the preceding transformations, if any.
 - e) 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 20 – *GWQ2* can not lack a <window clause>, since the syntactic transformation of Subclause 6.3, “<windowed table function>”, will create one if there is not one in *GWQ* already.
 - f) Let *NI* be the number of <set function specification>s directly contained in *GWQ2*.
 - g) Let *SFS_i*, $1 \text{ (one)} \leq i \leq NI$, be an enumeration of the <set function specification>s directly contained in *GWQ2*.
 - h) Let *SFSI_i*, $1 \text{ (one)} \leq i \leq NI$, be a list of <identifier>s that are distinct from each other and distinct from all <identifier>s contained in *GWQ2*.

- i) If $N1 = 0$ (zero), then let $SFSL$ be a zero-length string; otherwise, let $SFSL$ be:
- $$SFSL \text{ AS } SFSI_1, SFSI_2 \text{ AS } SFSI_2, \dots, SFSI_{N1} \text{ AS } SFSI_{N1}$$
- j) Let $HCNEW$ be obtained from HC by replacing each <set function specification> SFS_i by the corresponding <identifier> $SFSI_i$.
- k) Let $N2$ be the number of <column reference>s directly contained in SL or WIC .
- l) Let CR_j , 1 (one) $\leq j \leq N2$, be an enumeration of the <column reference>s directly contained in SL or WIC .
- m) 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 $SFSI_i$.
- n)
- o) If $N2 = 0$ (zero), then let $SFSL$ be a zero-length string; otherwise, let CRL be:
- $$CRL \text{ AS } CR_2 \text{ AS } CRI_2, \dots, CR_{N2} \text{ AS } CRI_{N2}$$
- p) Let $N3$ be the number of <derived column>s simply contained in SL that do not specify <as clause>.
- q) 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.
- r) Let $SL2$ be obtained from SL by replacing each <derived column> $DCOL_k$ by
- $$DCOL_k \text{ AS } COLN_k$$
- s) Let $GWQN$ be an arbitrary <identifier>.
- t) Let $SLNEW$ be the <select list> obtained from $SL2$ by replacing each <set function specification> SFS_i by $GWQN.SFSI_i$ and replacing each directly contained <column reference> CR_j by $GWQN.CRI_j$.
- u) 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 $GWQN.CRI_j$.
- v) If either $SFSL$ or CRL is a zero-length string, then let $COMMA$ be a zero-length string; otherwise, let $COMMA$ be “,” (a <comma>).
- w) GWQ is equivalent to the following <query specification>:

SC32 N00379 — FPDAM 9075:1999/AM1

7.6 <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 <windowed table function> that specifies ROW_NUMBER or whose associated <window specification> specifies ROWS.
- 5) 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 be one of the following:
 - a) Functionally dependent on G .
 - b) Contained in a <parameterized set function>.
 - c) Contained in the <within group specification> of an <ordered set function>.
- 6) Insert after SR 16)a)viii) A <windowed table function> whose <windowed table 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

No additional General Rules.

Conformance Rules

- 1) Replace CR 3) Without Feature T301, “Functional dependencies”, if T is a grouped table, then in each <value expression> contained in the <select list> of T , each column reference that references a column of T shall be one of the following:
 - a) A reference to a grouping column.
 - b) Contained in a <parameterized set function>.
 - c) Contained in the <within group specification> of an <ordered set function>.

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>
    | <general set function>
    | <binary set function>

<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

<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>

```

8.1 <aggregate function>

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 <windowed table function>.
 - b) Let DT be the declared type of the <value expression>.
 - c) If AF specifies a <general set function> whose <set qualifier> is DISTINCT and DT is a user-defined type, then the comparison form of DT shall be FULL.
 - d) If AF specifies a <set function type> that is MAX or MIN and DT is a user-defined type, then the comparison form of DT shall be FULL.
 - e) If AF specifies a <set function type> that is MAX or MIN, then DT shall not be a <collection type>, row type, reference type, or large object string type.
 - f) If EVERY, ANY, or SOME is specified, then DT shall be boolean and the declared type of the result is boolean.
 - g) If MAX or MIN is specified, then the declared type of the result is DT .
 - h) 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*.
 - i) 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*.
 - j) *STDDEV_POP(X)* is equivalent to *SQRT(VAR_POP(X))*.
 - k) *STDDEV_SAMP(X)* is equivalent to *SQRT(VAR_SAMP(X))*.
 - l) 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".
- 8) If <binary set function> is specified, then:
- a) The <dependent variable expression> *DVE* and the <independent variable expression> *IVE* shall not contain a <windowed table 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*.

Access Rules

None.

General Rules

- 1) If *COUNT(*)* is specified, then the result is the cardinality of *T*.
- 2) 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 *T* and eliminating null values. If one or more null values are eliminated, then a completion condition is raised: *warning — null value eliminated in set function*.
 - 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*.

8.1 <aggregate function>

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.

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 $(S2 - S1 * S1 / N) / (N - 1)$

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

- a) If <binary set function type> is specified, then 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 T 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 21 – 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 22 – 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 $\text{SQRT}(\text{POWER}(N * SUMXY - SUMX * SUMY, 2) / ((N * SUMX2 - SUMX * SUMX) * (N * SUMY2 - SUMY * SUMY)))$. 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 23 – 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.

8.1 <aggregate function>

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

NOTE 24 – 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 25 – 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

Case:

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

NOTE 26 – 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 $(SUMY * SUMX2 - SUMX * SUMXY) / (N * SUMX2 - SUMX * SUMX)$. 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.

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 of a structured 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>.

8.2 <sort specification list>

Function

Specify a sort order.

Format

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

```
<sort specification> ::=
    <sort key> [ <collate clause> ]
    [ <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>.
 - a) *DT* shall not be large object string, reference type, or array type.
 - b) If the <sort specification> contains a <collate clause>, then *DT* shall be character string.
 - c) If *DT* is a user-defined type, then the comparison form of *DT* shall be FULL.
- 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 , $1 \text{ (one)} \leq i \leq N$, 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>.
- 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 functions", conforming SQL language shall not specify <null ordering>.

9 Schema definition and manipulation

9.1 <drop routine statement>

Function

Destroy an SQL-invoked routine.

Format

!! No additional Format items

Syntax Rules

- 1) Replace Syntax Rule 4) If *SR* is the ordering function included in the user-defined descriptor of any user-defined type *UDT*, then:
 - a) Let *P* be a <predicate> that is dependent on *SR*, let *SFS* be a <set function specification> that is dependent on *SR*, let *OF* be a <windowed table function> that is dependent on *SR*, and let *GBC* be a <group by clause> that is dependent on *SR*.
NOTE 27 – The notion of a <predicate>, <set function specification>, <windowed table function>, or <group by clause> that is dependent on an SQL-invoked routine is defined in Subclause 4.4, “SQL-invoked routines”.
 - b) If RESTRICT is specified, then neither *P*, *SFS*, *OF*, nor *GBC* shall be contained in any of the following:
 - i) The <SQL routine body> of any routine descriptor.
 - ii) The <query expression> of any view descriptor.
 - iii) The <search condition> of any constraint descriptor or assertion descriptor.
 - iv) The triggered action of any trigger descriptor.
NOTE 28 – If CASCADE is specified, then such referencing objects will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

No additional Access Rules

General Rules

No additional General Rules

SC32 N00379 — FPDAM 9075:1999/AM1
9.1 <drop routine statement>

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 after SR 4) Let SK be any <sort key> whose declared type is UDT that is simply contained in a <window order clause> contained in a <windowed table function>.
- 2) Replace SR 5) If RESTRICT is specified, then neither P nor SK shall be contained in any of the following:
 - a) The <SQL routine body> of any routine descriptor.
 - b) The <query expression> of any view descriptor.
 - c) The <search condition> of any constraint descriptor or assertion descriptor.
 - d) The triggered action of any trigger descriptor.

NOTE 29 – If CASCADE is specified, then such referencing objects will be dropped as specified in the General Rules of this Subclause.

Access Rules

No additional Access Rules.

General Rules

- 1) Replace GR 1) Let R be any SQL-invoked routine that contains P or SK in its <SQL routine body>. Let SN be the specific name of R . The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SN CASCADE
```

- 2) Replace GR 2) Let V be any view that contains P or SK in its <query expression>. Let VN be the <table name> of V . The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW VN CASCADE
```

- 3) Replace GR 3) Let T be any table that contains P or SK in the <search condition> of any constraint C whose constraint descriptor included in the table descriptor of T . Let TN be the <table

SC32 N00379 — FPDAM 9075:1999/AM1

9.2 <drop user-defined ordering statement>

name> of *T*. Let *TCN* be the <constraint name> of *C*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE TN DROP CONSTRAINT TCN CASCADE
```

- 4) Replace GR 4) Let *A* be any assertion that contains *P* or *SK* in its <search condition>. Let *AN* be the <constraint name> of *A*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION AN CASCADE
```

- 5) Replace GR 5) Let *D* be any domain that contains *P* or *SK* in the <search condition> of any constraint descriptor or in the <default option> included in the domain descriptor of *D*. Let *DN* be the <domain name> of *D*. The following <drop domain statement> is effectively executed without further Access Rule checking:

```
DROP DOMAIN DN CASCADE
```

- 6) Replace GR 6) Let *T* be any trigger that contains *P* or *SK* in its triggered action. Let *TN* be the <trigger name> of *T*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER TN CASCADE
```

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_PRECEDING_OR_FOLLOWING_SIZE_IN_OLAP_FUNCTION:  
    constant SQLSTATE_TYPE := "22013";
```


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 30 – The deleted definitions are now placed in Subclause 8.2, “<sort specification list>”.

Syntax Rules

- 1) Replace SR 18)f)i)2)A)II) *SL* shall not specify SELECT DISTINCT or directly contain one or more <set function specification>s.
- 2) Delete SR 19)
- 3) Delete SR 22)

NOTE 31 – 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.

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 ) ;
```

SC32 N00379 — FPDAM 9075:1999/AM1

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
X	<i>All alternatives from ISO/IEC 9075-2</i> <i>All alternatives from ISO/IEC 9075-5</i> data exception	22	invalid preceding or following size in OLAP function	013

15 Conformance

15.1 General conformance requirements

Insert this paragraph The list of features that are implied by other features is shown in Table 3, “Implied feature relationships”.

Table 3—Implied feature relationships

Feature ID	Feature Description	Implied Feature ID	Implied Feature Description
T612	Advanced OLAP functions	T611	Elementary OLAP functions

Annex A **(informative)**

SQL conformance summary

Replicated paragraph The contents of this Annex summarizes all Conformance Rules, ordered by Feature ID and by Subclause.

- 1) To Be Supplied
- 2) Specifications for Feature , “”:

Annex B (informative)

Implementation-defined elements

Insert this paragraph This Annex references those features that are identified in the body of this amendment to ISO/IEC 9075 as implementation-defined.

Insert this paragraph The term *implementation-defined* is used to identify characteristics that may differ between SQL-implementations, but shall be defined for each particular SQL-implementation.

- 1) Insert this list element Subclause 4.2.2, “Windowed table functions”:
 - a) If PERCENT_RANK is specified, then the declared type of the result is approximate numeric with implementation-defined precision.
 - b) If CUME_DIST is specified, then the declared type of the result is approximate numeric with implementation-defined precision.
- 2) Insert this list element Subclause 6.1, “<numeric value function>”:
 - a) The declared type of the result of <natural logarithm> is approximate numeric with implementation-defined precision.
 - b) The declared type of the result of <exponential function> is approximate numeric with implementation-defined precision.
 - c) The declared type of the result of <power function> is approximate numeric with implementation-defined precision.
 - d) The declared type of the result of <floor function> is exact numeric with implementation-defined precision and scale 0 (zero).
 - e) The declared type of the result of <ceiling function> is exact numeric with implementation-defined precision and scale 0 (zero).
- 3) Insert this list element Subclause 8.1, “<aggregate function>”:
 - a) If COUNT is specified, then the declared type of the result is exact numeric with implementation-defined precision and scale of 0 (zero).
 - b) If SUM or AVG is specified, then:
 - i) If SUM is specified and the declared type of the argument is exact numeric with scale *S*, then the declared type of the result is exact numeric with implementation-defined precision and scale *S*.

- ii) If *AVG* is specified and the declared type of the argument 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*.
- iii) If the declared type of the argument is approximate numeric, then the declared type of the result is approximate numeric with implementation-defined precision not less than the precision of *DT*.
- iv) If the declared type of the argument is interval, then the declared type of the result is interval with the same precision as *DT*.
- c) 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*.
- d)
- e) If <binary set function type> is specified, then:
 - i) If *REGR_COUNT* is specified, then the declared type of the result is exact numeric with implementation-defined precision and scale of 0 (zero).
 - ii) If *REGR_AVGX* is specified, then:
 - 1) If the declared type *DTIVE* of the second parameter is exact numeric, then the declared type of the result is exact numeric with implementation-defined precision not less than the precision of *DTIVE* and implementation-defined scale not less than the scale of *DTIVE*.
 - 2) Otherwise, the declared type of the result is approximate numeric with precision not less than the precision of *DTIVE*.
 - iii) If *REGR_AVGY* is specified, then:
 - 1) If the declared type *DTDVE* of the first parameter is exact numeric, then the declared type of the result is exact numeric with implementation-defined precision not less than the precision of *DTDVE* and implementation-defined scale not less than the scale of *DTDVE*.
 - 1) Otherwise, the declared type of the result is approximate numeric with precision not less than the precision of *DTDVE*.
 - iv) Otherwise, the declared type of the result is approximate numeric with implementation-defined precision.
- 4) Insert this list element Subclause 8.2, “<sort specification list>”:
 - a) 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>.

Annex C (informative)

Implementation-dependent elements

Insert this paragraph This Annex references those places where this amendment to ISO/IEC 9075 states explicitly that the actions of a conforming SQL-implementation are implementation-dependent.

Insert this paragraph The term *implementation-dependent* is used to identify characteristics that may differ between SQL-implementations, but that are not necessarily specified for any particular SQL-implementation.

- 1) Insert this list element Subclause 4.3.1, “Windowed tables”:
 - a) The window name of a window defined implicitly by an <in-line window specification> is implementation-dependent.
- 2) Insert this list element Subclause 7.5, “<window clause>”:
 - a) If the <window order clause> of a <window specification> is a zero-length string, then the ordering is entirely implementation-dependent.
 - b) The ordering of peer rows within a partition is implementation-dependent, but the ordering shall be the same for all window structure descriptors that are order-equivalent. It shall also be the same for windows $W1$ and $W2$ if $W1$ is the ordering window for $W2$.
- 3) Insert this list element Subclause 7.6, “<query specification>”:
 - a) The <column name> of a <derived column> that is not a <column reference> and that has no <as clause> is implementation-dependent, but shall not be equivalent to the <column name> of any column, other than itself, of a table referenced by any <table reference> contained in the SQL-statement.
- 4) Insert this list element Subclause 8.1, “<aggregate function>”:
 - a) If the declared type of the argument of MAX or MIN is a user-defined type and the comparison of two values results in unknown, then the maximum or minimum is implementation-dependent.
- 5) Insert this list element Subclause 8.2, “<sort specification list>”:
 - a) 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.

- b) The relative ordering of two rows that are not distinct with respect to the <sort specification> is implementation-dependent.

Annex D (informative)

SQL feature and package taxonomy

This Annex describes a taxonomy of features of the SQL language.

Table 4, “SQL/OLAP feature taxonomy for features outside Core SQL”, contains a taxonomy of the features of the SQL language that are specified in this amendment to ISO/IEC 9075. In this table, the first column contains a counter that may be used to quickly locate rows of the table; these values otherwise have no use and are not stable — that is, they are subject to change in future editions of or even Technical Corrigenda to ISO/IEC 9075 without notice.

The second column, “Feature ID”, specifies the formal identification of each feature and each subfeature contained in the table. The Feature ID is stable and can be depended on to remain constant. A Feature ID value comprises either a letter and three digits or a letter, three digits, a hyphen, and one or two additional digits. Feature ID values containing a hyphen and additional digits indicate “subfeatures” that help to define complete features, which are in turn indicated by Feature ID values without a hyphen. Only entire features are used to specify the contents of Core SQL and various packages.

The “Feature Name” column contains a brief description of the feature or subfeature associated with the Feature ID value.

Table 4—SQL/OLAP feature taxonomy for features outside Core SQL

	Feature ID	Feature Name
1	T611	Elementary OLAP functions
2	T612	Advanced OLAP functions

Index

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

— A —

<aggregate function> • 23, 24, 25, 28, 30, 31, 50, **51**,
52, 57
Amendment 1 • 5
<asterisk> • 48, 51

— B —

<binary set function> • 24, **51**, 53, 56
<binary set function type> • **51**, 54, 57, 80

— C —

CEIL • 15, 19, 72
CEILING • 15, 19, 72
<ceiling function> • 9, **19**, 20, 21, 22, 79
<collate clause> • 38, 41, 58
<column reference> • 10, 32, 38, 39, 49, 81
<comma> • 19, 23, 38, 49, 51, 58
<computational operation> • 23, **51**
CORR • 15
COVAR_POP • 12, 15, 51, 55
COVAR_SAMP • 12, 15, 51, 55
CUME_DIST • 10, 13, 15, 27, 28, 29, 31, 79

— D —

data exception • 20, 21, 26, 42, 43, 44, 45, 46, 54, 73
DENSE_RANK • 10, 13, 15, 24, 28, 29
dependent • 12, 13, 14, 24, 25, 26, 30, 36, 39, 42,
45, 46, 49, 50, 51, 53, 54, 55, 56, 57, 59, 61,
81, 82
<dependent variable expression> • 12, 24, **51**, 54, 55,
56, 57

— E —

EXCLUDE • 15, 39, 46
<existing window name> • **38**, 40, 41, 42, 47
EXP • 15, 19, 21, 72
<exponential function> • 9, **19**, 20, 22, 79

— F —

Feature T611 • 31, 32, 46, 59
Feature T612 • 17, 22, 27, 31, 33, 46, 47
FLOOR • 15, 19, 72
<floor function> • 9, **19**, 20, 21, 22, 79
FOLLOWING • 15, 29, 39, 40, 42, 44, 45, 46, 65
<from clause> • 30, 33, 35, 41, 48

— G —

<general set function> • 23, 24, **51**, 52, 53, 54, 56, 57
<group by clause> • 30, 33, 41, 48, 61
grouped, windowed query • 48
grouped table aggregate functions • 11, 12
grouping operation • 9, 10
<grouping operation> • 23

— H —

<having clause> • 9, 30, 33, 36, 41, 48

— I —

<identifier> • 17, 48, 49
implementation-defined • 20, 24, 29, 52, 53, 55, 56,
58, 72, 79, 80
implementation-dependent • 13, 25, 26, 30, 42, 45,
46, 49, 54, 59, 81, 82
<independent variable expression> • 12, 24, **51**, 53,
54, 55, 56, 57
<in-line window specification> • **28**, 30, 48, 81
invalid preceding or following size in OLAP function •
42, 43, 44, 45, 46, 73
<inverse distribution function> • **23**, 24, 26, 27
<inverse distribution function argument> • **23**, 24, 26
<inverse distribution function type> • **23**

— L —

<left paren> • 19, 23, 28, 38, 51
LN • 15, 19, 21, 72

— N —

<natural logarithm> • 9, **19**, 20, 22, 79
 <new window name> • **38**, 39, 40, 41
 <nonparenthesized value expression primary> • **32**
 <non-reserved word> • **15**
 <null ordering> • 41, **58**, 59, 80
 NULLS • 13, 15, 43, 44, 45, **58**, 59
 null value eliminated in set function • 53, 54
 <numeric value expression base> • **19**, 21
 <numeric value expression exponent> • **19**, 21
 <numeric value function> • **19**
 numeric value out of range • 20, 21, 26, 54

— O —

OLAP • 13, 17, 22, 27, 28, 31, 32, 33, 42, 43, 44, 45, 46, 47, 59, 65, 73, 75, 83
 <ordered set function> • **23**, 36, 37, 39, 47, 50
 order-equivalent • 41, 42, 81
 <ordering specification> • 27, 41, 43, 44, 45, **58**, 67
 ordering window • 14, 41, 42, 81
 OTHERS • 15, 39, 46
 OVER • 15, 26, 27, 28, 29, 30

— P —

Part 1 • 3
 Part 2 • 3
 Part 3 • 3
 Part 4 • 3
 Part 5 • 3
 PARTITION • 15, 38
 partitioning column • 39, 42
 peers • 10, 14, 42, 43, 44, 45, 46, 59
 PERCENTILE_CONT • 13, 15, 23, 25, 26, 27
 PERCENTILE_DISC • 13, 15, 23, 27
 PERCENT_RANK • 10, 13, 15, 28, 29, 31, 79
 POWER • 15, 19, 20, 55, 56, 72
 <power function> • 9, **19**, 20, 22, 79
 precede • 10, 42, 43, 45, 59, 72
 PRECEDING • 15, 29, 38, 39, 40, 42, 44, 45, 46, 65

— R —

RANGE • 13, 15, 29, 38, 40, 42
 RANK • 10, 15, 24, 28, 29
 rank functions • 10
 <rank function type> • 23, 25, **28**, 50
 REGR_AVGX • 12, 15, 51, 55, 80
 REGR_AVGY • 12, 15, 51, 55, 80
 REGR_COUNT • 12, 15, 51, 53, 55, 80
 REGR_INTERCEPT • 12, 15, 51, 56
 REGR_R2 • 12, 15, 51, 55
 REGR_SLOPE • 12, 15, 51, 56
 REGR_SXX • 12, 15, 51, 55
 REGR_SXY • 12, 15, 51, 55
 REGR_SYY • 12, 15, 51, 55
 <reserved word> • **15**
 <right paren> • 19, 23, 28, 38, 51
 row number function • 10
 ROW_NUMBER • 10, 15, 26, 27, 28, 29, 50

— S —

<set function specification> • **23**, 24, 32, 48, 49, 52, 61, 67
 <set function type> • 23, **51**, 52, 56
 <set quantifier> • 23, 29, 48, **51**, 52
 sort direction • 59
 <sort key> • 14, 24, 25, 26, 29, 40, 41, 43, 44, **58**, 63, 67, 69
 <sort specification> • 24, 25, 26, 41, **58**, 59, 67, 82
 <sort specification list> • 23, 24, 25, 38, 42, **58**, 67, 80
 SQRT • 15, 19, 53, 55, 72
 <square root> • 9, **19**, 20, 22
 STDDEV_POP • 12, 15, 51, 52, 53, 57
 STDDEV_SAMP • 12, 15, 51, 52, 53, 57

— T —

<table expression> • 9, 13, 28, 30, **33**, 39, 41, 48
 TIES • 15, 39, 46

— U —

UNBOUNDED • 15, 29, 38, 39, 40, 42, 44, 45, 46
 <unordered set function> • **23**
 <unsigned value specification> • 32, 39, 40, 42, 43, 44, 45, 46

— V —

<value expression> • 11, 12, 13, 14, 23, 24, 25, 26, 29, 32, 50, 51, 52, 53, 56, 57, 58
 VAR_POP • 11, 12, 15, 51, 52, 53, 54, 55, 57, 80
 VAR_SAMP • 12, 15, 51, 52, 53, 54, 55, 57, 80

— W —

warning • 53, 54
 <what-if set function> • **23**, 24, 27
 <what-if value expression list> • **23**, 24, 25
 <where clause> • 30, 33, 35, 41, 48
 <width bucket bound 1> • **19**, 21
 <width bucket bound 2> • **19**, **20**, 21
 <width bucket count> • **19**, **20**, 21
 <width bucket function> • **19**, 20, 21, 22
 <width bucket operand> • **19**, 21
 WIDTH_BUCKET • 15, 19, 72
 window • 6, 7, 9, 10, 11, 13, 14, 17, 24, 25, 28, 29, 30, 31, 32, 33, 34, 35, 36, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 52, 53, 61, 63, 69, 79, 81
 <window clause> • 9, 25, 30, 33, **38**, 39, 40, 41, 42, 46, 47, 48, 49
 <window definition> • 13, **38**, 39, 40, 41
 <window definition list> • **38**
 <windowed table function> • 14, 24, **28**, 29, 30, 31, 32, 34, 35, 36, 39, 41, 48, 50, 52, 53, 61, 63
 <windowed table function type> • **28**, 30
 <window frame between> • 38, **39**, 40
 <window frame bound 1> • **39**, 40, 42
 <window frame bound 2> • **39**, 40, 42
 <window frame bound> • **39**
 <window frame clause> • 14, **38**, 40, 42

SC32 N00379 — FPDAM 9075:1999/AM1

- <window frame exclusion> • 14, 38, **39**
- <window frame extent> • **38**, 40
- <window frame following> • **39**, 40, 43, 44, 45, 46, 69
- <window frame preceding> • 38, **39**, 40, 42, 44, 45, 46, 69
- <window frame start> • **38**, 39, 40
- <window frame units> • **38**
- <window name> • **17**, 28, 29, 30, 31, 38, 40
- <window name or specification> • **28**, 30
- <window order clause> • 14, **38**, 39, 40, 41, 42, 43, 44, 45, 63, 81
- <window partition clause> • 13, 14, **38**, 39, 40, 41
- <window partition column reference> • **38**, 41
- <window partition column reference list> • **38**
- <window specification> • 28, 30, **38**, 46, 50, 69, 81
- <window specification details> • 29, **38**
- window structure descriptor • 10, 11, 14, 28, 33, 40, 41, 42, 52, 81
- <within group specification> • **23**, 24, 36, 37, 39, 47, 50

