INTERNATIONAL STANDARD

ISO/IEC 9075-16

First edition
2023-06

Information technology — Database languages SQL —

Part 16:
Property Graph Queries (SQL/PGQ)

Reference number
ISO/IEC 9075-16:2023(E)

Contents

Page

Tables

Table **Page**

Foreword

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

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs).

ISO and IEC draw attention to the possibility that the implementation of this document may involve the use of (a) patent(s). ISO and IEC take no position concerning the evidence, validity or applicability of any claimed patent rights in respect thereof. As of the date of publication of this document, ISO and IEC have not received notice of (a) patent(s) which may be required to implement this document. However, implementers are cautioned that this may not represent the latest information, which may be obtained from the patent database available at www.iso.org/patents and https://patents.iec.ch. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

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

This first edition of ISO/IEC 9075-16 is designed to be used in conjunction with the following editions of other parts of the ISO/IEC 9075 series, all published in 2023:

— ISO/IEC 9075-1, sixth edition;

— ISO/IEC 9075-2, sixth edition;

— ISO/IEC 9075-3, sixth edition;

— ISO/IEC 9075-4, seventh edition;

— ISO/IEC 9075-9, fifth edition;

— ISO/IEC 9075-10, fifth edition;

— ISO/IEC 9075-11, fifth edition;

— ISO/IEC 9075-13, fifth edition;

— ISO/IEC 9075-14, sixth edition;

— ISO/IEC 9075-15, second edition.

A list of all parts in the ISO/IEC 9075 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

Introduction

The organization of this document is as follows:

1) Clause 1, “Scope”, specifies the scope of this document.
2) Clause 2, “Normative references”, identifies additional standards that, through reference in this document, constitute provisions of this document.
3) Clause 3, “Terms and definitions”, defines the terms and definitions used in this document.
4) Clause 4, “Concepts”, presents concepts used in the definition of property graph queries.
5) Clause 5, “Lexical elements”, defines a number of lexical elements used in the definition of property graph queries.
6) Clause 6, “Scalar expressions”, defines a number of scalar expressions used in the definition of property graph queries.
7) Clause 7, “Query expressions”, defines the elements of the language that produce rows and tables of data as used in property graph queries.
8) Clause 8, “Predicates”, defines the predicates of the language.
9) Clause 9, “Additional common rules”, specifies additional rules for implicit or explicit invocation from other places in this document.
10) Clause 10, “Additional common elements”, defines additional common elements used in the definition of property graph queries.
11) Clause 11, “Schema definition and manipulation”, defines the schema definition and manipulation statements associated with the definition of property graph queries.
12) Clause 12, “Access control”, defines facilities for controlling access to SQL-data.
13) Clause 13, “SQL-client modules”, defines the facilities for using property graph queries.
14) Clause 14, “Diagnostics management”, defines the diagnostics management facilities.
15) Clause 15, “Information Schema”, defines viewed tables that contain schema information.
16) Clause 16, “Definition Schema”, defines base tables on which the viewed tables containing schema information depend.
17) Clause 17, “Status codes”, defines SQLSTATE values related to property graph queries.
18) Clause 18, “Conformance”, defines the criteria for conformance to this document.
19) Annex A, “SQL conformance summary”, is an informative Annex. It summarizes the conformance requirements of the SQL language.
20) Annex B, “Implementation-defined elements”, is an informative Annex. It lists those features for which the body of this document states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or other aspect is partly or wholly implementation-defined.
21) Annex C, “Implementation-dependent elements”, is an informative Annex. It lists those features for which the body of this document states that the syntax, the meaning, the returned results, the effect on SQL-data and/or schemas, or other aspect is partly or wholly implementation-dependent.

22) Annex D, “SQL optional feature taxonomy”, is an informative Annex. It identifies features of the SQL language specified in this document by an identifier and a short descriptive name. This taxonomy is used to specify conformance.

23) Annex E, “Deprecated features”, is an informative Annex. It lists features that the responsible Technical Committee intends not to include in a future edition of this document.

24) Annex F, “Incompatibilities with ISO/IEC 9075:2016”, is an informative Annex. It lists incompatibilities with the previous edition of this document.

25) Annex G, “Defect Reports not addressed in this edition of this document”, is an informative Annex. It describes the Defect Reports that were known at the time of publication of this document. Each of these problems is a problem carried forward from the previous edition of the ISO/IEC 9075 series. No new problems have been created in the drafting of this edition of this document.

In the text of this document, in Clause 5, “Lexical elements”, through Clause 18, “Conformance”, Subclauses begin new pages. Any resulting blank space is not significant.

Information technology — Database language SQL —

Part 16:
Property Graph Queries (SQL/PGQ)

1 Scope

This document defines ways for the SQL language to represent property graphs and to interact with them.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

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

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

ISO/IEC 9075-11, *Information technology — Database languages — SQL — Part 11: Information and Definition Schemas (SQL/Schemata)*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 9075-1, ISO/IEC 9075-2, and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at `https://www.iso.org/obp`

— IEC Electropedia: available at `https://www.electropedia.org/`

3.1
destination vertex
vertex (3.37) that is distinguished as the destination of a *directed edge* (3.4)

3.2
destination vertex table
table that contains data representing the *destination vertices* (3.1) of a *directed edge* (3.4)

3.3
defined label set
<pure property graph> distinguished set of labels

Note 1 to entry: Subclause 4.5.2, "Pure property graph", specifies six kinds of defined label sets: vertex defined label set, edge defined label set, edge triplet defined label set, source vertex defined label set, destination vertex defined label set, and endpoint vertex defined label set.

3.4
directed edge
edge (3.6) that distinguishes one of its *endpoints* (3.14) as its *source vertex* (3.30) and one of its endpoints as its *destination vertex* (3.1)

Note 1 to entry: A directed edge expresses a relationship that is asymmetric.

Note 2 to entry: The antonym is *undirected edge* (3.35).

3.5
directed graph
graph in which every *edge* (3.6) is directed

Note 1 to entry: The antonym is *undirected graph* (3.36).

3.6
edge
relationship
connection between two *vertices* (3.37)

Note 1 to entry: Both terms, "edge" and "relationship", are used in the real world to denote the same concept. Without judgement or prejudice, this document uses only the term "edge". In BNF productions, wherever the keyword EDGE is allowed, the keyword RELATIONSHIP can equally well be used instead.

3.7
edge pattern
path pattern (3.24) that matches a single *edge* (3.6)

3.8
edge table
table that contains data representing *edges* (3.6)

3.9
edge table alias
identifier that uniquely identifies an *edge table* (3.8) within an *SQL-property graph* (3.32)

3.10
edge variable
element variable (3.13) that is declared in an *edge pattern* (3.7) or as the second *iterator variable* (3.18) in <one row per step>

Note 1 to entry: An edge variable may be bound to a list of *edges* (3.6).

3.11
element pattern
edge pattern (3.7) or *vertex pattern* (3.38)

3.12
element table
edge table (3.8) or *vertex table* (3.39)

3.13
element variable
variable that may be bound to a list of *graph elements* (3.15)

Note 1 to entry: An element variable is either a *primary variable* (3.27) or an *iterator variable* (3.18).

Note 2 to entry: An element variable is either a *vertex variable* (3.41) or an *edge variable* (3.10).

3.14
endpoint
one of the two *vertices* (3.37) connected by an *edge* (3.6)

Note 1 to entry: Both endpoints of an edge may be the same vertex.

3.15
graph element
vertex (3.37) or *edge* (3.6)

3.16
graph pattern
set of one or more *path patterns* (3.24)

3.17
graph pattern variable
path variable (3.25), *subpath variable* (3.34), or *element variable* (3.13)

3.18
iterator variable
element variable (3.13) that is declared in <one row per iteration>

3.19
label
identifier associated with a *vertex* (3.37) or *edge* (3.6)

3.20
label expression
expression composed from *label* (3.19) names using disjunction, conjunction, and negation

Note 1 to entry: Disjunction, conjunction, and negation are denoted respectively by a vertical bar "|", ampersand "&" and exclamation mark "!", with parentheses for grouping. See Subclause 10.8, "<label expression>" for more details.

3.21
mixed graph
graph that allows both directed and undirected *edges* (3.6)

3.22
multigraph
graph that allows more than one *edge* (3.6) connecting two *vertices* (3.37)

3.23
path
sequence of an odd number of *graph elements* (3.15)

Note 1 to entry: A path always starts and ends with a *vertex* (3.37) and alternates between vertices and *edges* (3.6), such that each edge resides in the path between its *endpoints* (3.14). See Subclause 4.7.2, "Paths", for a fuller discussion of paths.

Note 2 to entry: A path may consist of a single vertex.

Note 3 to entry: A vertex or an edge may be contained multiple times in a path, including via self-loops.

3.24
path pattern
pattern (3.26) that matches a *path* (3.23)

3.25
path variable
graph pattern variable (3.17) that may be bound to a path binding that is matched by a *path pattern* (3.24)

Note 1 to entry: The extracted path of the path binding is a *path* (3.23).

Note 2 to entry: Extracted path and path binding are defined in Subclause 9.6, "Machinery for graph pattern matching".

3.26
pattern
syntax for specifying requirements on the bindings of variables

3.27
primary variable
element variable (3.13) that is declared in an *element pattern* (3.11)

3.28
property
<edge or vertex> (property name, data type) pair

Note 1 to entry: The value of a property is a value of its data type.

3.29
property graph
data structure consisting of zero or more *vertices* (3.37) and zero or more *edges* (3.6)

3.30
source vertex
vertex (3.37) that is distinguished as the source of a *directed edge* (3.4)

3.31
source vertex table
table that contains data representing the *source vertices* (3.30) of a *directed edge* (3.4)

3.32
SQL-property graph
SQL-schema object that represents *vertices* (3.37) and *edges* (3.6) using tables

3.33
subpath
path (3.23) fully contained in another path

Note 1 to entry: A subpath may be identical to its containing path.

3.34
subpath variable
graph pattern variable (3.17) that is declared at the head of a <parenthesized path pattern expression>

3.35
undirected edge
edge (3.6) that does not distinguish between its *endpoints* (3.14)

Note 1 to entry: An undirected edge expresses a relationship that is necessarily symmetric.

Note 2 to entry: The antonym is *directed edge* (3.4).

3.36
undirected graph
graph in which every *edge* (3.6) is undirected

Note 1 to entry: The antonym is *directed graph* (3.5).

3.37
vertex
node
fundamental unit of which a *property graph* (3.29) is formed

Note 1 to entry: plural: vertices or nodes.

Note 2 to entry: Both terms, “vertex” and “node”, are used in the real world to denote the same concept. Without judgement or prejudice, this document uses only the term “vertex”. In BNF productions, wherever the keyword VERTEX is allowed, the keyword NODE can equally well be used instead.

3.38
vertex pattern
path pattern (3.24) that matches a single *vertex* (3.37)

3.39
vertex table
table that contains data representing *vertices* (3.37)

3.40
vertex table alias
identifier that uniquely identifies a *vertex table* (3.39) within an *SQL-property graph* (3.32)

3.41
vertex variable
element variable (3.13) that is declared in a *vertex pattern* (3.38) or as an *iterator variable* (3.18) in <one row per vertex> or as the first or third iterator variable in <one row per step>

Note 1 to entry: A vertex variable may be bound to a list of *vertices* (3.37).

4 Concepts

This Clause modifies Clause 4, “Concepts”, in ISO/IEC 9075-2.

4.1 Notations and conventions

This Subclause modifies Subclause 4.1, “Notations and conventions”, in ISO/IEC 9075-2.

4.1.1 Notations

This Subclause modifies Subclause 4.1.1, “Notations”, in ISO/IEC 9075-2.

The notations used in this document are defined in ISO/IEC 9075-1.

The syntax defined in this document is available from the ISO website as a “digital artifact”. The syntax defined in this document is available from the ISO website as a “digital artifact”. See `https://standards.iso.org/iso-iec/9075/-16/ed-1/en/` to download digital artifacts for this document. To download the syntax defined in a plain-text format, select the file named `ISO_IEC_9075-16(E)_PGQ.bnf.txt`. To download the syntax defined in an XML format, select the file named `ISO_IEC_9075-16(E)_PGQ.bnf.xml`.

4.2 Columns, fields, and attributes

This Subclause modifies Subclause 4.15, “Columns, fields, and attributes”, in ISO/IEC 9075-2.

Insert after the 5th paragraph: The nullability characteristic of a column of <graph table> is defined by the Syntax Rules of Subclause 7.1, “<table reference>”.

4.3 SQL-statements

This Subclause modifies Subclause 4.41, “SQL-statements”, in ISO/IEC 9075-2.

4.3.1 SQL-statements classified by function

This Subclause modifies Subclause 4.41.2, “SQL-statements classified by function”, in ISO/IEC 9075-2.

4.3.1.1 SQL-schema statements

This Subclause modifies Subclause 4.41.2.1, “SQL-schema statements”, in ISO/IEC 9075-2.

Insert into the 1st paragraph, after the last list item:

— <property graph definition>.

— <alter property graph statement>.

— <drop property graph statement>.

4.4 Basic security model

This Subclause modifies Subclause 4.42, “Basic security model”, in ISO/IEC 9075-2.

4.4.1 Privileges

This Subclause modifies Subclause 4.42.2, "Privileges", in ISO/IEC 9075-2.

Insert into the 1st paragraph, after the last list item:

— An SQL-property graph.

Insert after the 8th paragraph: A privilege descriptor with an object that is an SQL-property graph *PG* and an action of SELECT is called a *property graph privilege descriptor* and identifies the existence of a privilege on *PG*.

4.5 SQL-property graphs

4.5.1 Introduction to SQL-property graphs

An SQL-property graph is described by an SQL-property graph descriptor. This descriptor includes the following.

— The schema-qualified name of the SQL-property graph.

— A pure property graph descriptor describing the pure property graph.

— A tabular property graph descriptor describing the tabular property graph.

The data making up an SQL-property graph is the data exposed through the vertex and edge tables that are part of the tabular property graph.

4.5.2 Pure property graph

A pure property graph is a mixed multigraph and consists of the following.

— A set of vertices. Each vertex consists of the following.

- An identifier that is unique within the pure property graph.

  NOTE 1 — The value of a vertex identifier is implementation-dependent (UV021) and is not required to be accessible. This unique identifier is used for definitional purposes to establish the identity of the vertex.

- A set of zero or more labels (also known as a *vertex defined label set*) that determines the properties of the vertex. A label has a name (i.e., an <identifier>).

- Zero or more properties. For each property:

  — its name;

  — its declared type, which can be any SQL data type,

  — its value.

— A set of edges. Each edge consists of the following.

- An identifier that is unique within the pure property graph.

  NOTE 2 — The value of an edge identifier is implementation-dependent (UV022) and is not required to be accessible. This unique identifier is used for definitional purposes to establish the identity of the edge.

- A set of zero or more labels (also known as an *edge defined label set*) that determines the properties of the edge. A label has a name (i.e., an <identifier>).

- Zero or more properties. For each property:

  — its name;

  - its declared type, which can be any SQL data type;
  - its value.
- An indication, whether the edge is directed or undirected.

  Case:

  - If the edge is directed, then an ordered pair of vertices, as follows.
    - A source vertex.
    - A destination vertex.

      NOTE 3 — The direction of an edge is from its source vertex to its destination vertex.

      NOTE 4 — Both the destination vertex and the source vertex of an edge are part of the same pure property graph as the edge.

      NOTE 5 — The source vertex and the destination vertex can be the same vertex.
  - If the edge is undirected, then an unordered pair of vertices, the endpoints of the undirected edge.

    NOTE 6 — The pure property graph model supports undirected edges, while the tabular graph model does not. An SQL-implementation can provide implementation-defined means to populate an undirected or mixed graph. This document specifies query syntax (<edge pattern> in Subclause 10.6, "<path pattern expression>") and semantics (the General Rules regarding <edge pattern> in Subclause 9.7, "Evaluation of a <path pattern expression>") to query undirected edges, if an SQL-implementation supports undirected edges.

    NOTE 7 — The two endpoints of an undirected edge can be the same vertex.

    NOTE 8 — The case of a directed edge whose source and destination vertex are the same is distinguished from an undirected edge whose endpoints are the same.

A pure property graph is described by a *pure property graph descriptor*. This descriptor includes the following.

- A set of zero or more labels. Each label consists of the following.
  - An <identifier> that is unique within the pure property graph descriptor.

    NOTE 9 — The <identifier> of a label is also known as the name of the label.
  - A set of zero or more properties. Each property consists of the following.
    - Its name, which is an <identifier> that is unique within the label.
    - Its declared type.
- A possibly empty set of vertex defined label sets.
- A possibly empty set of edge defined label sets.
  - For each edge defined label set, an indication whether edges that have that edge defined label set are always directed, always undirected, or possibly directed or undirected.
- A possibly empty set of *edge triplets*. An edge triplet consists of the following.
  - An *edge triplet defined label set*, which is a subset of an edge defined label set.
  - An indication whether the edge triplet describes directed edges or undirected edges.

    Case:

    - If the edge triplet describes directed edges, then:
      - a *source vertex defined label set*, which is a subset of a vertex defined label set;

        - a *destination defined vertex label set*, which is a subset of a vertex defined label set.

    — If the edge triplet describes undirected edges, then:

        - a pair of *endpoint vertex defined label sets*, each of which is a subset of a vertex defined label set.

4.5.3 Tabular property graph

A *tabular property graph* comprises the following.

— Zero or more vertex tables.

— Zero or more edge tables.

A tabular property graph *TPG* is described by a tabular property graph descriptor. This descriptor includes the following.

— The descriptors of zero or more vertex tables. Each vertex table descriptor includes the following.

- If the vertex table is based on a persistent base table or viewed table, then the schema-qualified name of that table *V*; otherwise, a <query expression> defining the vertex table.
- The vertex table alias that identifies *V* as a vertex table within the SQL-property graph that contains *TPG*.
- A list of columns that uniquely identify a row in *V*.
- One or more labels. Each label consists of the following.

    — An <identifier>.

    — A set of zero or more properties. Each property consists of the following.

        - A name, which is an <identifier> that is unique within the label.
        - A declared type.
        - A <value expression> associated with the property.

    NOTE 10 — While a pure property graph allows a vertex to have zero labels, in a tabular property graph a vertex has at least one label.

— The descriptors of zero or more edge tables. Each edge table descriptor includes the following.

- If the edge table is based on a persistent base table or viewed table, then the schema-qualified name of that table *E*; otherwise, a <query expression> defining the edge table.
- The edge table alias that identifies *E* as an edge table within the SQL-property graph that contains *TPG*.
- A list of columns that uniquely identify a row in *E*.
- One or more labels. Each label consists of the following.

    — An <identifier>.

    — A set of zero or more properties. Each property consists of the following.

        - A name, which is an <identifier> that is unique within the label.
        - A declared type.
        - A <value expression> associated with the property.

NOTE 11 — While a pure property graph allows an edge to have zero labels, in a tabular property graph an edge has at least one label.

- The name of the *source vertex table*.
- The *edge source key*: one or more columns (of the edge table) referencing one or more rows in the source vertex table.
- The *source vertex key*: one or more columns (of the source vertex table) that correspond to the edge source key
- The name of the *destination vertex table*.
- The *edge destination key*: one or more columns (of the edge table) referencing one or more rows in the destination vertex table.
- The *destination vertex key*: one or more columns (of the destination vertex table) that correspond to the edge destination key.

4.6 Operations involving property graphs

<graph table> is a kind of <table primary> that applies a <graph pattern> to a property graph and formats the results as a table.

<directed predicate> is a predicate to test whether an edge variable is bound to a directed edge.

NOTE 12 — <directed predicate> is specified for use in mixed graphs; however, this document does not provide a means to create a mixed graph.

<labeled predicate> is a predicate to test whether the graph element bound to an element variable satisfies a <label expression>.

<source/destination predicate> is a pair of predicates, to test whether a vertex variable is bound to the source or destination of the edge that is bound to an edge variable.

<all_different predicate> is a predicate to test whether a list of element variables are bound to pairwise distinct graph elements.

<same predicate> is a predicate to test whether a list of element variables are bound to the same graph element.

<bound predicate> is a predicate to test whether an element variable is bound to at least one graph element. Optional syntax of <bound predicate> tests whether an iterator variable is bound to the same position in a path binding as a primary variable.

<property_exists predicate> is a predicate to determine if the graph element bound to a singleton element variable has a particular property.

<element_id function> is a function to obtain a value that encapsulates the identity of a graph element. The declared type of <element_id function> is an implementation-defined (IV120) type that is suitable for equality operations and grouping operations. The value returned by <element_id function> encapsulates the identity of the graph element for at least the duration of the innermost executing SQL-statement.

<graphical match number function> is a function to obtain a unique implementation-dependent (UV016) number identifying the current match being returned by <graph table>.

<graphical path name function> is a function to obtain the name of the path variable of the path that the iterator variable(s) are currently iterating over.

<graphical element number function> is a function to obtain the 1-relative sequential number of the graph element in a path binding that an iterator variable is positioned on.

A *within-match aggregate* is a <set function specification> contained in a <graph table> without an intervening <query expression>. A within-match aggregate computes a value from the group of graph elements referenced by an element reference whose degree of reference is effectively bounded group.

4.7 Graph pattern matching

4.7.1 Summary of graph pattern matching

Graph pattern matching is the process of applying a list of special-purpose regular expressions called a <graph pattern> on a pure property graph *PG* in order to return a set of reduced matches. Each reduced match is a list of path bindings; each path binding is a function that maps the symbols in a word of a regular language to a path of *PG*. The regular language and related concepts such as path binding and reduced match are specified in Subclause 9.6, "Machinery for graph pattern matching", and Subclause 10.4, "<graph pattern>".

A <graph pattern> is a list of <path pattern>s. Each <path pattern> in the list is matched to *PG* to detect a possibly-empty set of path bindings in *PG* that correspond to the <path pattern>.

The cross-product of these sets is reduced by natural joins over those global singleton element variables that are exposed by each <path pattern> and that are bound to the same graph element in *PG*. The remaining tuples are called reduced matches; the set of these reduced matches may be empty, but cannot be infinite because of syntactic restrictions to guard against infinite cycling.

The behavior of the graph pattern matching is defined in this document.

NOTE 13 — A more detailed summary can be found in Subclause 4.7.6, "Path pattern matching".

Additional qualifying parameters (predicates, a selective <path search prefix>, a <path mode>, and the <different edges match mode>) that restrict the result may be supplied.

If *PG* contains cycles, then a match to a <path pattern> having an unbounded quantifier might return an infinite set of paths. However, this possibility is prevented by Syntax Rules that require the use of selective <path search prefix>s, restrictive <path mode>s, or the <different edges match mode> (or any combination thereof) to prevent infinitely-sized result sets.

4.7.2 Paths

A *path P* is a sequence of *n* graph elements of a property graph *PG*, such that:

— *n* is an odd number;

— *n* = 1 (one), then *P* has no edges;

— the graph element at each odd index is a vertex and the graph element at each even index is an edge that connects the pair of vertices immediately before and after the edge in the sequence.

A path contains a non-empty sequence of vertices and a (possibly empty) sequence of edges. If there are two or more vertices, then the path is a sequence of graph elements that starts with a vertex and is followed by a sequence of ordered (edge, vertex) pairs.

If *PG* is a multigraph, then an edge in the path between a pair of vertices is one of possibly several edges between those vertices in *PG*.

Every edge *E* in the path has an orientation. If *E* is undirected, then the orientation of *E* is *undirected*. Let *V1* be the vertex that immediately precedes *E* in the path, and let *V2* be the vertex that immediately follows *E*. If the source of *E* is *V1* and the destination of *E* is *V2*, then the orientation of *E* is *left to right*. It is also said that the orientation of *E* is *directed pointing right*. If the source of *E* is *V2* and the destination of *E* is *V1*, then the orientation of *E* is *right to left*. It is also said that the orientation of *E* is *directed pointing left*.

NOTE 14 — A directed self-edge (i.e., when *E* is directed and *V1* and *V2* are the same vertex), is oriented both left to right and right to left.

If no edge from *PG* appears more than once in a path, then the path is called a *trail*. If no vertex from *PG* appears more than once in a path, except possibly as the first and last vertices of the path, then the path is called *simple*. If no vertex from *PG* appears more than once in a path, then the path is called *acyclic*.

NOTE 15 — The term "path" is used in more than one way in mathematical graph theory and in informal technical presentations and discussions. In this document, the term path always denotes what a graph-theoretic work could call a partially-oriented walk in a pure property graph. Such a graph is a mixed multigraph: edges can be directed or undirected, and there can be multiple edges between two vertices.

4.7.3 Path patterns

A <path pattern> is an expression built from the following syntactic elements, governed by the Format and Syntax Rules of Subclause 10.4, "<graph pattern>", and other Subclauses.

— An optional <path variable declaration>, to declare a path variable to be bound to a path binding.

— An optional <path pattern prefix>.

NOTE 16 — <path pattern prefix> is described in Subclause 4.7.7, "Path modes", and Subclause 4.7.8, "Selective path search prefixes", and specified in Subclause 10.5, "<path pattern prefix>".

— A mandatory <path pattern expression>.

A <path pattern expression> is an expression built recursively from <element pattern>s, governed by the Format and Syntax Rules of Subclause 10.6, "<path pattern expression>", and other Subclauses.

An <element pattern> is a pattern to match a single graph element. There are two kinds of <element pattern>s.

— <vertex pattern>:

A <vertex pattern> is a pattern to match a single vertex. A <vertex pattern> comprises at a minimum a matching pair of parentheses, which may contain optional <element pattern filler>, described subsequently.

— <edge pattern>:

An <edge pattern> is a pattern to match a single edge. An <edge pattern> is either a <full edge pattern> (which optionally permits <element pattern filler>) or an <abbreviated edge pattern> (which does not support <element pattern filler>). These two major classes of <edge pattern> have seven variants each, for the seven possible non-empty combinations of the three edge orientations (the individual edge orientations being directed pointing left, undirected, or directed pointing right). Thus there are fourteen varieties of <edge pattern>.

<element pattern filler> provides three optional components of <vertex pattern>s and <full edge pattern>s.

— <element variable declaration>, to declare an element variable to be bound to a graph element by the <element pattern>.

— <label expression>, a predicate regarding the labels of the graph element that is bound by the <element pattern>. A <label expression> is an expression formed from <label name>s and the <wildcard label> "%", using the operation signs <vertical bar> "|" for disjunction, <ampersand> "&" for conjunction, <exclamation mark> "!" for negation, and balanced pairs of parentheses for grouping.

— <element pattern where clause>, a <search condition> to be satisfied by the graph element that is bound by the <element pattern>.

<path pattern expression>s are regular expressions built recursively from <element pattern>s using the following operations, governed by the Format and Syntax Rules of Subclause 10.6, "<path pattern expression>", and other Subclauses.

— concatenation, indicated syntactically by string concatenation (i.e., no operation sign).

NOTE 17 — <element pattern>s and more complex <path pattern expression>s can be concatenated in ways that appear to juxtapose either two <vertex pattern>s or two <edge pattern>s. These topologically inconsistent patterns are understood during pattern matching as follows.

- Two consecutive <vertex pattern>s bind to the same vertex.
- Two consecutive <edge pattern>s conceptually have an implicit <vertex pattern> between them.

— Grouping, using matching pairs of parentheses to form a <parenthesized path pattern expression>. A <parenthesized path pattern expression> may optionally contain the following.

- A <subpath variable declaration> to declare a subpath variable.
- A <search condition> to constrain matches.

— Alternation, indicated by <vertical bar> or <multiset alternation operator>.

NOTE 18 — Alternation with <vertical bar> provides set semantics using deduplication of redundant equivalent reduced matches, whereas alternation with <multiset alternation operator> provides multiset semantics, with no deduplication.

— Quantification, indicated by a postfix <graph pattern quantifier>, which may be affixed to an <edge pattern> or a <parenthesized path pattern expression>.

— <questioned path primary>, indicated by a postfix <question mark> affixed to an <edge pattern> or a <parenthesized path pattern expression>.

NOTE 19 — Unlike the situation in many regular expression languages, the <question mark> operator in <path pattern expression>s is not the same as {0,1}, the difference being that <questioned path primary> exposes its singleton element or subpath variables as conditional singletons, whereas {0, 1}, in common with all other quantifiers, exposes all element or subpath variables as a group.

4.7.4 Graph pattern variables

A *graph pattern variable GPV* is a site identified by an <identifier> (the *name* of the graph pattern variable) and having a value determined by a multi-path binding *MPB* to a <graph pattern>.

NOTE 20 — "Site" is defined in ISO/IEC 9075-1 Subclause 4.7.1, "Sites".

There are four kinds of graph pattern variables.

— Vertex variables; the value of a vertex variable is a list of vertices.

— Edge variables; the value of an edge variable is a list of edges.

— Path variables; the value of a path variable is a path binding.

— Subpath variables.

NOTE 21 — Subpath variables are not bound to a value in this edition of this document. They serve to assure multiset semantics in <path multiset alternation>.

In a <graph table> *GT*, an <identifier> shall not identify more than one graph pattern variable; thus *GT* defines a namespace in which there is a one-to-one correspondence between the names of graph pattern variables and the graph pattern variables that they name.

NOTE 22 — Because of this one-to-one correspondence, certain terms that are defined for graph pattern variables are also defined for their names, so that the name of the variable and the variable itself can be used interchangeably in the rules. These terms include "declare" and "expose".

Vertex variables and edge variables are collectively called *element variables*. An element variable is declared in either an <element pattern> (in which case it is a *primary element variable*) or in a <one row per iteration> in which case it is an *iterator variable*).

A primary vertex variable *VV* and its name are declared by an <element variable declaration> simply contained in a <vertex pattern>.

A primary edge variable *EV* and its name are declared by an <element variable declaration> simply contained in a <full edge pattern>.

A primary variable may be declared in more than one <element pattern>. A multiply declared primary variable *MDPV* expresses a natural equijoin in two circumstances.

— If *MDPV* is declared in both operands of a <path concatenation>.

— If *MPDV* is declared in two or more <path pattern>s of a <graph pattern>.

NOTE 23 — Declaring an element variable in two operands of a <path pattern union> or <path multiset alternation> does not express a natural equijoin.

<one row per vertex> declares a single iterator vertex variable and its name.

<one row per step> declares an iterator vertex variable, an iterator edge variable, and another iterator vertex variable, as well as their names.

Iterator element variables cannot be multiply declared.

A path variable *PV* and its name are declared by a <path variable declaration> simply contained in a <path pattern>. A path variable may only be declared once.

A subpath variable *SV* and its name are declared by a <subpath variable declaration> simply contained in a <parenthesized path pattern expression>. More than one operand of a <path pattern union> may declare *SV*; otherwise *SV* cannot be multiply declared.

4.7.5 References to graph pattern variables

Graph pattern variables are visible within the <graph table> *GT* in which they are declared. They may be referenced in scalar expressions and <search condition>s within *GT*.

Iterator variables and path variables may only be referenced in <graph table column definition>s.

Primary variables may be referenced in the following BNF non-terminals.

— <element pattern where clause>.

— <parenthesized path pattern where clause>.

— <graph pattern where clause>.

— <graph table column definition>.

If a primary element variable *PEV* is declared in a <quantified path primary> *QPP*, then it may bind to more than one graph element. References to *PEV* are interpreted contextually: if the reference occurs outside *QPP*, then the reference is to the complete list of graph elements that are bound to *PEV*. In this circumstance *PEV* is said to have *group degree of reference*. If the reference does not cross a quantifier, then the reference has *singleton degree of reference* and references at most one graph element, even if the multi-path binding binds *PEV* multiple times.

NOTE 24 — For example,

```
(X) -[E WHERE E.P > 1]->{1,10} (Y) WHERE SUM(E.P) < 100
```

This example references primary edge variable E twice: once in the <edge pattern> and once outside the <edge pattern>. Within the <edge pattern>, E has singleton degree of reference and the <property reference> E.P references a property of a single edge. On the other hand, the reference within the SUM aggregate has group degree of reference (because of the quantifier {1,10}) and references the list of edges that are bound to E.

A reference *R* to a graph pattern variable *GPV* is termed *local* in the following circumstances.

— If *GPV* is declared in an <element pattern> *EP* and *R* is contained in the <element pattern where clause> of *EP*.

— If *GPV* is declared in a <parenthesized path pattern expression> *PPPE* and *R* is contained in the <parenthesized path pattern where clause> of *PPPE*.

— If *R* is in a <graph pattern where clause> or a <graph table column definition>.

R has a degree of reference, determined by the Syntax Rules of Subclause 10.10, "<element reference>" and Subclause 10.11, "<path reference>". The degree of reference of R is one of the following: unconditional singleton, conditional singleton, or effectively bounded group.

NOTE 25 — Subpath variables cannot be referenced by SQL language defined by this document, though it is possible they will be referenceable in a future edition. At present their only use is to distinguish operands of a <path multiset alternation>.

NOTE 26 — In general, an element variable that is declared within a <quantified path primary> is bound by a multi-path binding to a list of graph elements. The reference *R*, on the other hand, can reference a proper subset of this list, based on the syntactic context in which *R* is placed. The degree of reference expresses the cardinality of the list that *R* references, as follows: an unconditional singleton references a list of length 1 (one), a conditional singleton references a list of length 0 (zero) or 1 (one), an effectively bounded group references a finite list. Syntax rules prohibit the possibility of referencing an infinite list.

A reference to a path variable always has unconditional singleton degree of reference.

References to graph pattern variables are subject to the following constraints.

— An operand *OP* of <path pattern union> or <path multiset alternation> *U* may only reference element variables declared in *OP*, or outside of *U*.

— A non-local reference shall have singleton degree of reference.

— A group reference shall be contained in an aggregated argument of a <set function specification>. The group references in an aggregated argument of a <set function specification> shall reference the same graph pattern variable. All other references to graph pattern variables in a <set function specification> shall have singleton degree of reference.

— A selective <path pattern> *SPP* shall not reference a graph pattern variable that is not declared in *SPP*.

— Iterator variables shall only be referenced in <graph table column definition>s.

If *GT* has a <graph table export clause> other than EXPORT NO SINGLETONS, then certain singleton <graph pattern variable>s (the exported <graph pattern variable>s) are visible in the <query specification> whose <from clause> simply contains *GT*. A syntactic transformation defines an equivalent <query specification> in which the <graph table> specifies EXPORT NO SINGLETONS, and all references to exported <graph pattern variable>s have been placed in <graph table column definition>s.

4.7.6 Path pattern matching

Path pattern matching is performed by Subclause 10.4, "<graph pattern>", which in turn may invoke Subclause 9.7, "Evaluation of a <path pattern expression>", and Subclause 9.8, "Evaluation of a selective <path pattern>", as well as other Subclauses incidentally invoked for expression evaluation.

In more detail, a <path pattern> can be seen as a regular expression built from <vertex pattern>s and <edge pattern>s. To formalize this, an alphabet is formed (in Subclause 9.6, "Machinery for graph pattern matching"), comprising the element variable names, plus additional special symbols for the anonymous vertex symbol, the anonymous edge symbol, bracket symbols to indicate the beginning and ending of bindings to <parenthesized path pattern expression>s, and subpath symbols to mark the beginning and ending of subpaths. The precise specification of the corresponding regular language is found in Subclause 9.7, "Evaluation of a <path pattern expression>".

Each <path pattern> of a <graph pattern> is evaluated independently of each other, resulting in a set of path bindings. A path binding is a list of elementary bindings; each elementary binding is an ordered pair (*LET*, *GE*), where *LET* is a member of the alphabet, and *GE* is

Case:

— if *LET* is a vertex variable or the anonymous vertex symbol, then a vertex;

— if *LET* is an edge variable or the anonymous edge symbol, then an edge;

— Otherwise, equal to *LET*.

NOTE 27 — In the formal semantics, <label expression>s are evaluated at this stage, but <search condition>s are not; hence, it is possible that some path bindings will be subsequently rejected because they fail a <search condition>. Implementations are, of course, free to "push down" predicate evaluation provided that the ultimate results are the same as prescribed by the formal semantics.

Projecting the elementary bindings of a path binding to the first component yields a word of the regular language of the <path pattern>. Projecting to the second component yields an annotated path, which is a path interspersed with mark-up to indicate the beginning and ending of <parenthesized path pattern expression>s and the beginning and ending of subpaths.

If a <path pattern> has an unbounded quantifier that is not in the scope of a restrictive <path mode> or the <different edges match mode>, then there may be infinitely many path bindings. Such a <path pattern> must have a selective <path search prefix> *SPSP*. Subclause 9.8, "Evaluation of a selective <path pattern>", is invoked to reduce this potentially infinite set of path bindings to a finite set. All <search condition>s contained in the selective <path pattern> are evaluated to reduce the set of path bindings prior to making the final selection according to *SPSP*.

NOTE 28 — An implementation cannot generate an infinite set and then apply <search condition>s; instead, it must enumerate the search space in a fashion enabling it to arrive at the same result as specified by the formal semantics. The formal semantics do not specify the algorithm for this enumeration, only the result.

At this point, there is a finite set of path bindings for each <path pattern> in a <graph pattern>. The cross product of these sets is formed; a member of the cross product is called a multi-path binding. A multi-path binding does not violate any restrictive <path mode> or <different edges match mode> that may be in force. The set of multi-path bindings is reduced by enforcing natural equijoins on the unconditional singleton variables exposed by the <path pattern>s.

Then, the <search condition>s in the <graph pattern> are evaluated, potentially further reducing the set of multi-path bindings. Those that remain satisfy all the selective <path search prefix>es and all the <search condition>s of the <graph pattern>.

Then, the function *REDUCE* (defined in Subclause 9.6, "Machinery for graph pattern matching") is applied to the remaining multi-path bindings. *REDUCE* removes the elementary bindings of the bracket symbols, and collapses adjacent anonymous vertex symbol bindings into a single elementary binding. The results, now called reduced matches, are deduplicated according to set semantics.

NOTE 29 — If there is a <path pattern union> in the <graph pattern>, then duplicates can arise.

4.7.7 Path modes

A <path mode> may be specified for any <parenthesized path pattern expression> or any <path pattern> from the following choices.

— WALK, the default <path mode>, is the absence of any filtering implied by the other <path mode>s.

— TRAIL, where path bindings with repeated edges are not returned.

— ACYCLIC, where path bindings with repeated vertices are not returned.

— SIMPLE, where path bindings with repeated vertices are not returned unless these repeated vertices are the first and the last in the path.

Using trail, acyclic, or simple matching path modes for all unbounded quantifiers guarantees that the result set of a graph pattern matching will be finite.

NOTE 30 — There is no implied hierarchy between path modes. Specifically, path mode SIMPLE does not imply path mode TRAIL. While such an implication holds for directed graphs, it does not hold for graphs with undirected edges. For instance, the pattern ( ) - ( ) - ( ) on a graph with two vertices and one undirected edge between them returns a match under SIMPLE but does not return any matches under TRAIL.

4.7.8 Selective path search prefixes

The set of path bindings resulting from a graph pattern match can be further restricted by a selective <path search prefix> *SPSP*. *SPSP* is defined by partitioning the potentially infinite set of path bindings by the endpoints, which are the first and last vertices bound by the path bindings.

NOTE 31 — This partitioning is crucial to the definition of *SPSP*. *SPSP* makes a selection of some number of path bindings from each partition. For example, a path binding is "shortest" if its length is minimal within its partition. A "shortest" path binding in one partition can be longer than a "shortest" path binding in another partition.

SPSP can constrain the result set in the following ways.

— <any path search>: to non-deterministically pick some number of path bindings from each partition; the number is specified by a <simple value specification>.

— <all shortest path search>: to pick all the shortest path bindings in each partition.

— <counted shortest path search>: to non-deterministically pick some number of shortest path bindings from each partition; the number is specified by a <simple value specification>.

— <counted shortest group search>: to group each partition into groups of path bindings having the same length, order the groups by path length, and pick all path bindings in some number of groups from the front of each partition; the number is specified by a <simple value specification>.

The specification of *SPSP* guarantees that the result set of a graph pattern matching will be finite.

4.7.9 Match modes

A <graph pattern> *GP* may optionally specify a <match mode> that applies to all <path pattern>s simply contained in *GP*.

There are two <match mode>s.

— DIFFERENT EDGES: A matched edge is not permitted to bind to more than one edge variable. There are no restrictions on matched vertices.

— REPEATABLE ELEMENTS: There are no restrictions on matched edges or matched vertices.

If a <match mode> is not specified, then an implementation-defined (ID022) <match mode> is implicit.

5 Lexical elements

This Clause modifies Clause 5, “Lexical elements”, in ISO/IEC 9075-2.

5.1 <SQL terminal character>

This Subclause modifies Subclause 5.1, “<SQL terminal character>”, in ISO/IEC 9075-2.

Function

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

Format

```
<SQL special character> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <tilde>

<tilde> ::=
  ~ !! U+007E
```

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 <token> and <separator>

This Subclause modifies Subclause 5.2, "<token> and <separator>", in ISO/IEC 9075-2.

Function

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

Format

```
<delimiter token> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <bracket right arrow>
  | <bracket tilde right arrow>
  | <left arrow>
  | <left arrow bracket>
  | <left arrow tilde>
  | <left arrow tilde bracket>
  | <left minus right>
  | <left minus slash>
  | <left tilde slash>
  | <minus left bracket>
  | <minus slash>
  | <multiset alternation operator>
  | <right bracket minus>
  | <right bracket tilde>
  | <slash minus>
  | <slash minus right>
  | <slash tilde>
  | <slash tilde right>
  | <tilde left bracket>
  | <tilde right arrow>
  | <tilde slash>

<bracket right arrow> ::=
  ]-> !! <U+005D, U+002D, U+003E>

<bracket tilde right arrow> ::=
  ]~> !! <U+005D, U+007E, U+003E>

<left arrow> ::=
  <- !! <U+003C, U+002D>

<left arrow bracket> ::=
  <-[ !! <U+003C, U+002D, U+005B>

<left arrow tilde> ::=
  <~ !! <U+003C, U+007E>

<left arrow tilde bracket> ::=
  <~[ !! <U+003C, U+007E, U+005B>

<left minus right> ::=
  <-> !! <U+003C, U+002D, U+003E>

<left minus slash> ::=
  <-/ !! <U+003C, U+002D, U+002F>

<left tilde slash> ::=
  <~/ !! <U+003C, U+007E, U+002F>
```

```
<minus left bracket> ::=
  -[ !! <U+002D, U+005B>

<minus slash> ::=
  -/ !! <U+002D, U+002F>

<multiset alternation operator> ::=
  |+| !! <U+007C, U+002B, U+007C>

<right bracket minus> ::=
  ]- !! <U+005D, U+002D>

<right bracket tilde> ::=
  ]~ !! <U+005D, U+007E>

<slash minus> ::=
  /- !! <U+002F, U+002D>

<slash minus right> ::=
  /-> !! <U+002F, U+002D, U+003E>

<slash tilde> ::=
  /~ !! <U+002F, U+007E>

<slash tilde right> ::=
  /~> !! <U+002F, U+007E, U+003E>

<tilde left bracket> ::=
  ~[ !! <U+007E, U+005B>

<tilde right arrow> ::=
  ~> !! <U+007E, U+003E>

<tilde slash> ::=
  ~/ !! <U+007E, U+002F>

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

  | ALL_DIFFERENT

  | BINDING_COUNT

  | ELEMENT_ID | ELEMENT_NUMBER | EXPORT

  | GRAPH | GRAPH_TABLE

  | MATCHNUM

  | PATH_LENGTH | PATH_NAME | PROPERTY_EXISTS

  | SAME

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

  | ACYCLIC

  | BINDINGS | BOUND

  | DESTINATION | DIFFERENT | DIRECTED

  | EDGE | EDGES | ELEMENTS

  | LABEL | LABELED
```

```
    | NODE

    | PATHS | PROPERTIES | PROPERTY | PROPERTY_GRAPH_CATALOG | PROPERTY_GRAPH_NAME
    | PROPERTY_GRAPH_SCHEMA

    | RELATIONSHIP | RELATIONSHIPS

    | SHORTEST | SINGLETONS | STEP

    | TABLES | TRAIL

    | VERTEX

    | WALK

<edge synonym> ::=
    EDGE | RELATIONSHIP

<edges synonym> ::=
    EDGES | RELATIONSHIPS

<vertex synonym> ::=
    NODE | VERTEX
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

5.3 Names and identifiers

This Subclause modifies Subclause 5.4, “Names and identifiers”, in ISO/IEC 9075-2.

Function

Specify names.

Format

```
<destination vertex table alias> ::=
  <vertex table alias>

<edge table alias> ::=
  <identifier>

<element table name> ::=
  <table name>

<element table alias> ::=
  <identifier>

<graph pattern variable> ::=
    <element variable>
  | <path or subpath variable>

<path or subpath variable> ::=
    <path variable>
  | <subpath variable>

<element variable> ::=
  <identifier>

<label name> ::=
  <identifier>

<path variable> ::=
  <identifier>

<subpath variable> ::=
  <identifier>

<property graph name> ::=
  <schema qualified name>

<property name> ::=
  <identifier>

<source vertex table alias> ::=
  <vertex table alias>

<vertex table alias> ::=
  <identifier>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after the last GR: An <element variable> identifies an element variable, which may be bound to a list of graph elements.

2) Insert after the last GR: A <path variable> identifies a path variable, which is bound to a path binding that is matched by a <path pattern>.

3) Insert after the last GR: A <subpath variable> identifies a subpath variable.

4) Insert after the last GR: A <property name> identifies a property in an SQL-property graph.

5) Insert after the last GR: A <label name> identifies a label in an SQL-property graph.

6) Insert after the last GR: A <destination vertex table alias> identifies a destination vertex table in an SQL-property graph.

7) Insert after the last GR: An <edge table alias> identifies an edge table in an SQL-property graph.

8) Insert after the last GR: An <element table alias> identifies an element table in an SQL-property graph.

9) Insert after the last GR: An <element table name> identifies a table.

10) Insert after the last GR: A <property graph name> identifies an SQL-property graph.

11) Insert after the last GR: A <source vertex table alias> identifies a source vertex table in an SQL-property graph.

12) Insert after the last GR: A <vertex table alias> identifies a vertex table in an SQL-property graph.

Conformance Rules

No additional Conformance Rules.

6 Scalar expressions

This Clause modifies Clause 6, "Scalar expressions", in ISO/IEC 9075-2.

6.1 <value expression primary>

This Subclause modifies Subclause 6.3, "<value expression primary>", in ISO/IEC 9075-2.

Function

Specify a value that is syntactically self-delimited.

Format

```
<non-parenthesized value expression primary> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <property reference>
  | <graphical value expression primary>

<graphical value expression primary> ::=
    <element_id function>
  | <graphical match number function>
  | <graphical path name function>
  | <graphical element number function>
  | <graphical path length function>
```

Syntax Rules

1) Augment SR 1) by adding "<property reference> and <graphical value expression primary>" to the list of BNF non-terminals simply contained in <value expression primary>.

2) Insert after SR 1): The declared type of <graphical value expression primary> is the declared type of the simply contained <element_id function>, <graphical match number function>, <graphical path name function>, <graphical element number function>, or <graphical path length function>.

Access Rules

No additional Access Rules.

General Rules

1) Augment GR 1) by adding "<property reference> and <graphical value expression primary>" to the list of BNF non-terminals simply contained in <value expression primary>.

2) Insert after GR 1): The value of <graphical value expression primary> is the value of the simply contained <element_id function>, <graphical match number function>, <graphical path name function>, <graphical element number function>, or <graphical path length function>.

Conformance Rules

No additional Conformance Rules.

6.2 <identifier chain>

This Subclause modifies Subclause 6.6, "<identifier chain>", in ISO/IEC 9075-2.

Function

Disambiguate a period-separated chain of identifiers.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 7): A property *PR* is said to be *refinable* if the declared type of *PR* is a row type, a JSON type, or a structured type.

2) Augment the lead text of SR 9) by adding "a property *PR*" to the list of possible referents of a basis.

3) Augment SR 9)b)iv) by adding "<element variable> " as alternatives to both occurrences of <correlation name>.

4) Insert after SR 9)b)iv)1): If *N* = 2 and *EN* is an exposed <element variable>, then

   a) Let *GT* be the <graph table> that is the scope of *EN*. Let *SPG* be the SQL-property graph identified by the <graph reference> simply contained in *GT*, and let *GP* be the <graph pattern> simply contained in *GT*. Let *EV* be the element variable that is identified by *EN*.

   b) The Syntax Rules of Subclause 9.2, "Contextual inference of a set of labels", are applied with *SPG* as *SQL-PROPERTY GRAPH*, *GP* as *GRAPH PATTERN*, *EV* as *ELEMENT VARIABLE*, and the <graph table column definition>, <parenthesized path pattern where clause> or <graph pattern where clause> that simply contains *IC* as *CONTEXT*; let *SOL* be the *SET OF LABELS* returned from the application of those Syntax Rules.

   c) Let *SOPN* be the set of property names of the properties of the labels of *SOL*.

   d) If I_2 is a property name in *SOPN*, then PIC_2 is a candidate basis of *IC*, the scope of PIC_2 is the scope of *EN*, and the referent of PIC_2 is the property *PR* named by I_2.

5) Insert after SR 9)b)iv)2): If *N* > 2 and *EN* is an exposed <element variable>, then

   a) Let *GT* be the <graph table> that is the scope of *EN*. Let *SPG* be the SQL-property graph identified by the <graph reference> simply contained in *GT*, and let *GP* be the <graph pattern> simply contained in *GT*. Let *EV* be the element variable that is identified by *EN*.

   b) The Syntax Rules of Subclause 9.2, "Contextual inference of a set of labels", are applied with *SPG* as *SQL-PROPERTY GRAPH*, *GP* as *GRAPH PATTERN*, *EV* as *ELEMENT VARIABLE*, and the <graph table column definition>, <parenthesized path pattern where clause>, or <graph pattern where clause> that simply contains *IC* as *CONTEXT*; let *SOL* be the *SET OF LABELS* returned from the application of those Syntax Rules.

   c) Let *SOPN* be the set of property names of the properties of the labels of *SOL*.

   d) If I_2 is a property name in *SOPN* and the property *PR* named by I_2 is refinable, then PIC_2 is a candidate basis of *IC*, the scope of PIC_2 is the scope of *EN*, and the referent of PIC_2 is the property *PR* named by I_2.

6) Insert after SR 16): A <basic identifier chain> whose basis referent is a property is a *property reference*.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 4): If *BIC* is a property reference, then *BIC* references the property that is the basis referent of *BIC*.

Conformance Rules

No additional Conformance Rules.

6.3 <set function specification>

This Subclause modifies Subclause 6.9, "<set function specification>", in ISO/IEC 9075-2.

Function

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

Format

No additional Format items.

Syntax Rules

1) Insert after SR 3): An <element reference> simply contained in an <element variable count function> is an *aggregated argument* of *SFE*.

2) Insert after SR 4): If <set function specification> *SFS* is contained in <graph table> *GT* without an intervening <query expression>, then:

   a) <grouping operation> shall not be specified.

   b) *SFS* is a *within-match aggregate*.

   c) The aggregated arguments of *SFS* shall reference exactly one element variable *AEV* whose degree of reference is group. The degree of reference of *AEV* shall be effectively bounded group. *AEV* is called the *aggregated element variable* of *SFS*.

      NOTE 32 — The aggregated element variable can be referenced more than once; for example, SUM (P.X * P.Y) references the element variable P twice. Since P is the only element variable referenced in the aggregated argument of SUM, the degree of reference of P will be effectively bounded group.

   d) Every element variable other than *AEV* that is referenced in *SFS* shall have singleton degree of reference.

      NOTE 33 — Assuming that *P* and *Q* are element variables, in the example

      ```
      PERCENTILE_CONT (P.X) WITHIN GROUP (ORDER BY Q.Y)
      ```

      P.X is a non-aggregated argument and therefore P must have singleton degree of reference. Q.Y is an aggregated argument and therefore Q is the aggregated element variable and must have effectively bounded group degree of reference.

      As another example:

      ```
      SUM (P.X * Q.Y)
      ```

      one of the variables P and Q must have group degree of reference (thereby being the aggregated element variable) and the other must have singleton degree of reference.

3) Insert after SR 6)a): If *SFS* is a within-match aggregate, then the aggregation query of *SFS* is *GT*.

4) Convert SR 7) to be: Case:

   a) If *SFS* is a within-match aggregate, then *SFS* shall be contained in a <parenthesized path pattern where clause>, <graph pattern where clause> or <graph table columns clause> simply contained in *GT*

   b) Otherwise, the original SR 7)

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after the last CR: Without Feature G120, “Within-match aggregates”, conforming SQL language shall not contain a within-match aggregate.

6.4 <case expression>

This Subclause modifies Subclause 6.12, "<case expression>", in ISO/IEC 9075-2.

Function

Specify a conditional value.

Format

```
<case operand> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <element reference>

<when operand> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <directed predicate part 2>
  | <labeled predicate part 2>
  | <source predicate part 2>
  | <destination predicate part 2>
  | <bound predicate part 2>
```

Syntax Rules

1) Insert after SR 2)c): If any <when operand> is <directed predicate part 2>, <labeled predicate part 2>, <source predicate part 2>, <destination predicate part 2>, or <bound predicate part 2>, then *CO* shall be <element reference> and every <when operand> shall be <directed predicate part 2>, <labeled predicate part 2>, <source predicate part 2>, <destination predicate part 2>, or <bound predicate part 2>; otherwise, *CO* shall not be <element reference> and no <when operand> shall be <directed predicate part 2>, <labeled predicate part 2>, <source predicate part 2>, <destination predicate part 2>, or <bound predicate part 2>.

   NOTE 34 — This rule disambiguates whether a single <identifier> is interpreted as an <element reference> or a <row value predicand> that is a single <identifier>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

6.5 <property reference>

Function

Reference a property of a graph element.

Format

```
<property reference> ::=
  <basic identifier chain>
```

Syntax Rules

1) The <basic identifier chain> *BIC* simply contained in a <property reference> *PR* shall be a property reference.

2) The declared type of *PR* is the declared type of *BIC*.

3) Let *EV* be the first <identifier> simply contained in *BIC*. *EV* is an <element variable>. Let *ER* be the <element reference>:

   EV

   ER shall be valid according to the Syntax Rules of Subclause 10.10, "<element reference>".

Access Rules

None.

General Rules

1) Let *LOE* be the list of graph elements bound to *ER*, and let *PN* be the property name contained in *ER*.

   NOTE 35 — The list of graph elements bound to *ER* is determined by the General Rules of Subclause 9.9, "Applying bindings to evaluate an expression", which is invoked prior to evaluating a <value expression> or <search condition> that contains *PR*.

2) If the degree of reference of *ER* is singleton, then

   Case:

   a) If *LOE* is empty, then the result of *PR* is the null value.

   b) Otherwise, let *E* be the sole graph element in *LOE*.

      Case:

      i) If *E* does not have a property whose name is *PN*, then the result of *PR* is the null value.

      ii) If the property of *E* whose name is *PN* is flagged as unusable, then an exception condition *EC* is raised where *EC* is the exception condition recorded as additional information when the property was marked as unusable.

         NOTE 36 — The exception condition is recorded as additional information in GR 6)b)iii)3)F) of Subclause 9.5, "Converting a tabular property graph to a pure property graph".

      iii) Otherwise, the result of *PR* is the value of the property of *E* whose name is *PN*.

NOTE 37 — A <property reference> that references a singleton element variable is resolved by this Subclause. References to a group element variable must be contained in an aggregated argument of a <set function specification>. The <set function specification> contains an <aggregate function>; the General Rules of Subclause 10.1, "<aggregate function>", invoke the General Rules of Subclause 9.10, "Applying bindings to evaluate a subexpression of an aggregate", to decompose such group references into a list of singleton references, so this Subclause only needs to handle singletons.

Conformance Rules

1) Without Feature G090, "Property reference", conforming SQL language shall not contain a <property reference>.

6.6 <element_id function>

Function

Generate a unique identifier for a graph element.

Format

```
<element_id function> ::=
  ELEMENT_ID <left paren> <element reference> <right paren>
```

Syntax Rules

1) The <element reference> shall have singleton degree of reference.

2) The declared type of <element_id function> is an implementation-defined (IV120) type that is permitted as the declared type of an operand of an equality operation according to the Syntax Rules of Subclause 9.11, “Equality operations”, in ISO/IEC 9075-2 and as the declared type of an operand of a grouping operation according to the Syntax Rules of Subclause 9.12, “Grouping operations”, in ISO/IEC 9075-2.

Access Rules

1) Access Rules for <element_id function> are implementation-defined (IA033).

General Rules

1) Let *LOE* be the list of graph elements bound to the <element reference>.

2) Case:

   a) If *LOE* is empty, then the value of <element_id function> is the null value.

   b) Otherwise, let *GE* be the sole graph element in *LOE*. The value of <element_id function> is an implementation-dependent (UV017) value that encapsulates the identity of *GE* in the SQL-property graph that contains *GE* for the duration of the innermost executing SQL-statement.

Conformance Rules

1) Without Feature G100, “ELEMENT_ID function”, conforming SQL language shall not contain an <element_id function>.

6.7 <graphical match number function>

Function

Return a number that uniquely identifies a match in a set of matches.

Format

```
<graphical match number function> ::=
  MATCHNUM <left paren> <right paren>
```

Syntax Rules

1) <graphical match number function> *GMNF* shall be contained in the <graph table columns clause> of a <graph table> *GT*, or in the scope of the exported <graph pattern variable>s of a <graph table> *GT*.

2) The declared type of the result of *GMNF* is implementation-defined (IV121) exact numeric with scale 0 (zero).

Access Rules

None.

General Rules

1) The value of the result of *GMNF* is the current match number.

NOTE 38 — The current match number is specified in the General Rules of Subclause 7.1, "<table reference>".

Conformance Rules

1) Without Feature G803, "MATCHNUM function", conforming SQL language shall not contain <graphical match number function>.

6.8 <graphical path name function>

Function

Return the name of the path that the iterator variables are currently iterating over.

Format

```
<graphical path name function> ::=
  PATH_NAME <left paren> <right paren>
```

Syntax Rules

1) <graphical path name function> *GPNF* shall be contained in the <graph table columns clause> of a <graph table> *GT*, or in the scope of the exported <graph pattern variable>s of a <graph table> *GT*.

2) *GT* shall specify <one row per iteration>.

3) The declared type of the result of *GPNF* is character string with implementation-defined (IL005) maximum length, character set SQL_IDENTIFIER, and collation SQL_IDENTIFIER.

Access Rules

None.

General Rules

1) The value of the result of *GPNF* is

   Case:

   a) If there is a current path name, then a character string whose value is equivalent to the current path name.

   b) Otherwise, the null value.

   NOTE 39 — The current path name is specified by the General Rules of Subclause 9.11, "Applying bindings to generate a row".

Conformance Rules

1) Without Feature G800, "PATH_NAME function", conforming SQL language shall not contain <graphical path name function>.

6.9 <graphical element number function>

Function

Return the sequential element number of the graph element binding that an iterator variable is currently positioned on.

Format

```
<graphical element number function> ::=
  ELEMENT_NUMBER <left paren> <element reference> <right paren>
```

Syntax Rules

1) <graphical element number function> *GENF* shall be contained in the <graph table columns clause> of a <graph table> *GT*, or in the scope of the exported <graph pattern variable>s of a <graph table> *GT*.

2) *GT* shall specify <one row per iteration> *ORPI*, and <element reference> shall reference an iterator variable *IV* declared by *ORPI*.

3) The declared type of the result of *GENF* is implementation-defined (IV122) exact numeric with scale 0 (zero).

Access Rules

None.

General Rules

1) The value of the result of *GENF* is the current element number of *IV*.

NOTE 40 — The current element number of an iterator variable is specified by the General Rules of Subclause 9.11, "Applying bindings to generate a row".

Conformance Rules

1) Without Feature G801, "ELEMENT_NUMBER function", conforming SQL language shall not contain <graphical element number function>.

6.10 <graphical path length function>

Function

Return the length (number of edges) of a path binding or a subpath binding.

Format

```
<graphical path length function> ::=
  PATH_LENGTH <left paren> <path reference> <right paren>
```

Syntax Rules

1) <graphical path length function> *GPLF* shall be contained in the <graph table columns clause> of a <graph table> *GT*, or in the scope of the exported <graph pattern variable>s of a <graph table> *GT*.

2) The declared type of the result of *GPLF* is implementation-defined (IV123) exact numeric with scale 0 (zero).

Access Rules

None.

General Rules

1) Let *PB* be the path binding that is the value of the <path reference>.

2) The value of the result *GPLF* is

   Case:

   a) If *PB* contains any vertex bindings, then the number of edge bindings in *PB*.

   b) Otherwise, the null value.

Conformance Rules

1) Without Feature G802, “PATH_LENGTH function”, conforming SQL language shall not contain <graphical path length function>.

7 Query expressions

This Clause modifies Clause 7, "Query expressions", in ISO/IEC 9075-2.

7.1 <table reference>

This Subclause modifies Subclause 7.6, "<table reference>", in ISO/IEC 9075-2.

Function

Reference a table.

Format

```
<table primary> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <graph table derived table>

<graph table derived table> ::=
  <graph table> [ <correlation or recognition> ]

<graph table> ::=
  GRAPH_TABLE
      <left paren> <graph reference> MATCH <graph pattern> <graph table shape>
      <right paren>

<graph table shape> ::=
  [ <graph table rows clause> ]
  [ <graph table columns clause> ]
  [ <graph table export clause> ]

<graph table rows clause> ::=
    ONE ROW PER MATCH
  | <one row per iteration>

<one row per iteration> ::=
    <one row per vertex>
  | <one row per step>

<one row per vertex> ::=
  ONE ROW PER <vertex synonym>
      <left paren> <vertex variable> <right paren>
      [ <in paths clause> ]

<one row per step> ::=
  ONE ROW PER STEP <left paren> <vertex variable 1> <comma> <edge variable>
      <comma> <vertex variable 2> <right paren>
      [ <in paths clause> ]

<vertex variable> ::=
  <element variable>

<vertex variable 1> ::=
  <element variable>

<edge variable> ::=
  <element variable>

<vertex variable 2> ::=
  <element variable>

<in paths clause> ::=
  IN <left paren> <path variable> [ { <comma> <path variable> }... ] <right paren>

<graph table columns clause> ::=
  COLUMNS <left paren> <graph table column definition>
      [ { <comma> <graph table column definition> }... ] <right paren>

<graph table column definition> ::=
    <value expression> [ AS <column name> ]
  | <all properties reference>

<all properties reference> ::=
  <element reference> <period> <asterisk>

<graph table export clause> ::=
    EXPORT ALL SINGLETONS [ <export except> ]
  | EXPORT SINGLETONS
        <left paren> <export variable list> <right paren>
  | EXPORT NO SINGLETONS

<export except> ::=
  EXCEPT <left paren> <export variable list> <right paren>

<export variable list> ::=
  <graph pattern variable> [ { <comma> <graph pattern variable> }... ]
```

Syntax Rules

1) Augment SR 5) by adding "<graph table derived table>" to the list of <table primary>s that support lateral join.

2) Insert after SR 11): If *TP* simply contains <graph table derived table> *GTDT*, then:

   a) Let *GT* be the <graph table> simply contained in *GTDT*. Let *GP* be the <graph pattern> simply contained in *GT*.

   b) If *GT* contains another <graph table> *GT2*, then the name of every graph pattern variable exposed by *GP* shall be distinct from the name of every graph pattern variable exposed by the <graph pattern> simply contained in *GT2*.

   c) If *GT* does not simply contain <graph table export clause>, then

      Case:

      i) If *GT* simply contains <graph table columns clause>, then the <graph table export clause> EXPORT NO SINGLETONS is implicit.

      ii) Otherwise, the <graph table export clause> EXPORT ALL SINGLETONS is implicit.

         NOTE 41 — In this case a later syntactic transform will generate a <graph table columns clause> and change the <graph table export clause> to EXPORT NO SINGLETONS.

   d) If *GT* simply contains <graph table columns clause>, then *GTDT* shall simply contain <correlation or recognition>.

   e) Let *SPG* be the SQL-property graph identified by the <graph reference>.

   f) Case:

i) If *GT* does not simply contain <graph table rows clause>, then ONE ROW PER MATCH is implicit.

ii) If <graph table rows clause> specifies <one row per vertex> *ORPV*, then the <vertex variable> *VV* contained in *OPRV* shall not be equivalent to any <element variable>, <path variable>, or <subpath variable> declared in *GP*. *OPRV* declares *VV* and an iterator variable *IV* whose name is *VV*. *OPRV exposes VV* and *IV* as unconditional singelton.

iii) If <graph table rows clause> specifies <one row per step> *ORPS*, then <vertex variable 1> *VV1*, <edge variable> *EDGY*, and <vertex variable 2> *VV2* shall be pairwise distinct and shall not be equivalent to any <element variable>, <path variable>, or <subpath variable> declared in *GP*.

1) *ORPS* declares *VV1* and an iterator variable *IV1* whose name is *VV1*. *ORPS* exposes *VV1* and *IV1* as unconditional singleton.

2) *ORPS* declares *EDGY* and an iterator variable *IE* whose name is *EDGY*. *ORPS* exposes *EDGY* and *IE* as conditional singleton.

3) *ORPS* declares *VV2* and an iterator variable *IV2* whose name is *VV2*. *ORPS* exposes *VV2* and *IV2* as conditional singleton.

g) Iterator variables shall not be referenced in *GP*.

h) Case:

i) If <graph table rows clause> *GTRC* specifies <in paths clause> *IPC*, then every <path variable> contained in *IPC* shall be equivalent to a <path variable> declared by *GP*.

ii) If <graph table rows clause> *GTRC* specifies <one row per iteration> and does not specify <in paths clause>, then at least one of the following shall be true:

1) *GP* simply contains only one <path pattern>.

2) Every <path pattern> simply contained in GP declares a <path variable>.

NOTE 42 — This rule assures that the PATH_NAME function unambiguously determines which path pattern an iterator variable is iterating over.

i) For each <all properties reference> *APD* simply contained in the <graph table columns clause>:

i) Let *ER* be the <element reference> contained in *APD*. *ER* shall have singleton degree of reference. Let *EV* be the element variable referenced by *ER*.

ii) The Syntax Rules of Subclause 9.3, "Expansion of an <all properties reference>", are applied with *SPG* as *SQL-PROPERTY GRAPH*, *GP* as *GRAPH PATTERN*, and *EV* as *ELEMENT VARIABLE*; let *LOPR* be the *LIST OF PROPERTY REFERENCES* returned from the application of those Syntax Rules.

iii) Case:

1) If *LOPR* is the zero-length character string, then

Case:

A) If *APD* is preceded by a <comma>, then that <comma> and *APD* are deleted.

B) If *APD* is followed by a <comma>, then that <comma> and *APD* are deleted.

C) Otherwise, *APD* is deleted.

2) Otherwise, *APD* is replaced by *LOPR*.

j) If a <graph table column definition> *GTCD* does not contain a <column name>, then let *VE* be the <value expression> simply contained in *GTCD*. Exactly one of the following shall be true:

   i) *VE* is a <property reference> *PR*. In this case, the <column name> contained in *GTCD* is implicitly the <property name> contained in *PR*.

   ii) *VE* is a <column reference> *CR*. In this case, the <column name> contained in *GTCD* is implicitly the <column name> contained in *CR*.

k) If *GT* simply contains an explicit or implicit <graph table export clause> *GTEC* other than EXPORT NO SINGLETONS, then:

   i) *GTDT* shall not be the <table reference> simply contained in a <merge statement>.

   ii) Let *GTFC* be the <from clause> that simply contains *GTDT*, and let *GTQS* be the <query specification> that simply contains *GTFC*.

   iii) The <select list> simply contained in *GTQS* shall not be <asterisk>.

   iv) *GTQS* shall not contain another <graph table derived table>.

   v) *GTDT* shall not contain <row pattern recognition clause and name>.

   vi) Case:

      1) If *GTDT* simply contains a <correlation name>, then let *GTCN* be that <correlation name>.

      2) Otherwise, let *GTCN* be an implementation-dependent (UV018) <correlation name> that is not in conflict with any range variable of *GTFC*.

         NOTE 43 — Conflicts between range variables are defined in Subclause 7.5, "<from clause>", in ISO/IEC 9075-2.

   vii) Let *NS* be the number of distinct path variables, subpath variables, and element variables exposed by *GP* with singleton degree of exposure.

      NOTE 44 — "Degree of exposure" is defined in Subclause 10.4, "<graph pattern>" and Subclause 10.6, "<path pattern expression>".

   viii) Let SV_1, ..., SV_{NS} be an enumeration of the names of the distinct path variables, subpath variables, and element variables exposed by *GP* with singleton degree of exposure.

      NOTE 45 — "Degree of exposure" is defined in Subclause 10.4, "<graph pattern>" and Subclause 10.6, "<path pattern expression>".

   ix) If *GTEC* simply contains <export variable list> *EVS*, then let *NE* be the number of <graph pattern variable>s contained in *EVS*, and let EVV_1, ..., EVV_{NE} be the list of <graph pattern variable>s contained in *EVS*. For all *i*, 1 (one) $\leq i \leq NE$, EVV_i shall be a <graph pattern variable> exposed by *GP* with singleton degree of exposure. The <graph pattern variable>s EVV_1, ..., EVV_{NE} shall be pairwise distinct.

      NOTE 46 — "Degree of exposure" is defined in Subclause 10.4, "<graph pattern>" and Subclause 10.6, "<path pattern expression>".

   x) Case:

      1) If *GTEC* contains <export except>, then let *FSV* be the set difference of { SV_1, ..., SV_{NS} } minus { EVV_1, ..., EVV_{NE} }.

      2) If *GTEC* contains EXPORT ALL SINGLETONS, then let *FSV* be the set { SV_1, ..., SV_{NS} }.

      3) Otherwise, let *FSV* be the *ESV*.

xi) The <graph pattern variable>s contained in *FSV* are the *exported <graph pattern variable>s* of *GT*.

xii) An exported <graph pattern variable> *EEV* is *in conflict with* a range variable *RV* if at least one of the following is true:

1) *RV* is a <table name> and *EEV* is equivalent to the <qualified identifier> of *RV*.

2) *RV* is a correlation name and *EEV* is equivalent to *RV*.

xiii) An exported <graph pattern variable> *EEV* shall not be in conflict with a range variable of *GTFC*.

xiv) *GT* is transformed into a <graph table> whose <graph table export clause> is EXPORT NO SINGLETONS as follows:

1) Let *GTSC* be the scope clause of *GTFC*.

NOTE 47 — The scope clause of *GTFC* is defined in Subclause 7.5, "<from clause>", in ISO/IEC 9075-2.

2) The *scope* of the exported <graph pattern variable>s of *GT* comprises the following:

A) Every <table primary> that supports lateral join, is simply contained in *GTFC* and is preceded by *GT*.

B) The <select list>, <where clause>, <group by clause>, <having clause>, and <window clause> of *GTSC*.

C) If *GTSC* is the <query specification> that is the <query expression body> of a simple table query *STQ*, then the scope of the exported <graph pattern variable>s of *GT* also includes the <order by clause> of *STQ*.

3) Case:

A) If *GT* does not simply contain a <graph table columns clause>, then let *GTCOLS* be an empty list.

B) Otherwise, let *GTCOLS* be the list of <graph table column definition>s simply contained in the <graph table columns clause> simply contained in *GT*.

4) Let *APRS* be the set of every <all properties reference> contained in the scope of the exported <graph pattern variable>s. Let APR_1, ..., APR_{NAPR} be an enumeration of those <all properties reference>s.

For every *i*, 1 (one) ≤ *i* ≤ *NAPR*:

A) APR_i shall be contained in the <select list> *SL* of *GTQS*.

B) Let XV_i be the element variable referenced by APR_i.

C) The Syntax Rules of Subclause 9.3, "Expansion of an <all properties reference>", are applied with *SPG* as *SQL-PROPERTY GRAPH*, *GP* as *GRAPH PATTERN*, and XV_i as *ELEMENT VARIABLE*; let $LOPR_i$ be the *LIST OF PROPERTY REFERENCES* returned from the application of those Syntax Rules

D) Case:

I) If $LOPR_i$ is the zero-length character string, then

Case:

1) If APR_i is preceded by a <comma>, then that <comma> and APR_i are deleted.

2) If APR_i is followed by a <comma>, then that <comma> and APR_i are deleted.

3) Otherwise, APR_i is deleted.

II) Otherwise APR_i is replaced by $LOPR_i$.

NOTE 48 — As a result of the preceding rules, the <all properties reference>s have been replaced by a (possibly empty) list of <property reference>s.

5) Let *PRS* be the set of every <property reference> contained in the scope of the exported <graph pattern variable>s. Let PR_1, ..., PR_{NPRS} be an enumeration of those <property reference>s. Let $PRALIAS_1$, ..., $PRALIAS_{NPRS}$ be a list of implementation-dependent (UV019) <identifier>s that are pairwise distinct and also distinct from every explicit or implicit column name of the <graph table column definition>s contained in *GTCOLS*.

For every *i*, $1 \leq i \leq NPRS$:

A) Let PV_i be the <element variable> that is the first <identifier> contained in PR_i and let PN_i be the property name that is the second <identifier> contained in PR_i.

B) Case:

I) If PR_i is the <value expression> of a <derived column> that has no <as clause>, then PR_i is replaced by the <column reference>

```
GTCN.PRALIASi AS PNi
```

NOTE 49 — The preceding preserves the implicit column name of the <derived column>.

II) Otherwise, PR_i is replaced by

```
GTCN.PRALIASi
```

C) The following <graph table column definition> is appended to *GTCOLS*:

```
PVi.PNi AS PRALIASi
```

6) Let *GPPS* be the set of every <graph element predicate> and every <graphical value expression primary> contained in the scope of the exported <graph pattern variable>s. Let $OTHER_1$, ..., $OTHER_{NO}$ be an enumeration of those <graph element predicate>s and <graphical value expression primary>s. Let $OALIAS_1$, ..., $OALIAS_{NO}$ be a list of implementation-dependent (UV019) <identifier>s that are pairwise distinct and also distinct from every explicit or implicit column name of the <graph table column definition>s contained in *GTCOLS*.

NOTE 50 — This step is necessarily performed after the syntactic transformations in Subclause 6.12, "<case expression>", in ISO/IEC 9075-2 to convert <simple case>s to <searched case>s, which can construct some <graph element predicate>s.

For every *i*, 1 (one) $\leq i \leq NO$:

A) $OTHER_i$ is replaced by

```
GTCN.OALIAS_i
```

B) The following <graph table column definition> is appended to *GTCOLS*:

```
OTHER_i AS OALIAS_i
```

7) Let *NCOLS* be the number of <graph table column definition>s contained in *GTCOLS* after the preceding insertions. Let $COLDEF_1$, ..., $COLDEF_{NCOLS}$ be the list of <graph table column definition>s contained in *GTCOLS*.

8) The <graph table export clause> of *GT* is replaced by EXPORT NO SINGLETONS, and the <graph table columns clause> of *GT*, if any, is replaced by

```
COLUMNS ( COLDEF_1, ..., COLDEF_NCOLS )
```

l) After the preceding syntactic transformation, there shall be at least one <graph table column definition>.

m) The explicit or implicit <column name>s specified in the <graph table column definition>s shall be distinct from one another.

n) The degree of the table specified by *GT* is the number of <graph table column definition>s simply contained in *GT*. For each <graph table column definition> *GTCD* simply contained in *GT*, a column descriptor is determined, as follows:

i) The column name of the column is the explicit or implicit <column name> simply contained in *GTCD*.

ii) The declared type of the column is the declared type of the <value expression> *VE* simply contained in *GTCD*.

iii) Case:

1) If the SQL-implementation supports Feature T101, “Enhanced nullability determination”, and *VE* conforms to an implementation-defined (IE011) rule that enables the SQL-implementation to correctly infer that *VE* cannot be null, then the column’s nullability characteristic is known not null.

2) Otherwise, the column’s nullability characteristic is possibly nullable.

3) Augment the lead text of SR 12) by adding “<graph table>” to the list of BNF non-terminals simply contained in *TP*.

4) Augment SR 12)b)i)2) by adding “<graph table>” to the list of BNF non-terminals simply contained in *TP*.

NOTE 51 — That is, the output of <graph table> can supply the row pattern input table for row pattern recognition.

5) Insert after SR 21)a): If <table primary> specifies <graph table>, then the <table primary> is not generally updatable, not simply updatable, not effectively updatable, and not insertable-into.

Access Rules

1) Let *PG* be the SQL-property graph identified by the <graph reference>.

Case:

a) If *GT* is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include SELECT on *PG*.

b) Otherwise, the current privileges shall include SELECT on *PG*.

General Rules

1) Insert after GR 6): If *TP* simply contains <graph table>, then:

   a) Let *SPG* be the SQL-property graph identified by <graph reference>. The General Rules of Subclause 9.5, "Converting a tabular property graph to a pure property graph", are applied with *SPG* as *SQL-PROPERTY GRAPH*; let *G* be the *PURE PROPERTY GRAPH* returned from the application of those General Rules.

   b) Let *GP* be the <graph pattern> after the transformations in the Syntax Rules of other Subclauses. Let *PPL* be the <path pattern list> simply contained in *GP*.

   c) The General Rules of Subclause 9.6, "Machinery for graph pattern matching", are applied with *G* as *PURE PROPERTY GRAPH* and *PPL* as *PATH PATTERN LIST*; let *MACH* be the *MACHINERY* returned from the application of those General Rules.

   d) The following components of *MACH* are identified:

      i) *ABC*, the alphabet, formed as the disjoint union of the following:

         1) *SVV*, the set of names of vertex variables.

         2) *SEV*, the set of names of edge variables.

         3) *SPS*, the set of subpath symbols.

         4) *SAS*, the set of anonymous symbols.

         5) *SBS*, the set of bracket symbols.

      ii) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.

   e) The General Rules of Subclause 10.4, "<graph pattern>", are applied with *G* as *PURE PROPERTY GRAPH*, *GP* as *GRAPH PATTERN*, *PPL* as *PATH PATTERN LIST*, and *MACH* as *MACHINERY*; let *SM* be the *SET OF REDUCED MATCHES* returned from the application of those General Rules.

   f) Each reduced match *M* in *SM* is assigned a unique implementation-dependent (UV020) exact numeric value with scale 0 (zero), called the *match number* of *M*.

      NOTE 52 — The match numbers are arbitrary numbers and need not be assigned sequentially, since there is no presumption that the set of reduced matches is in any order.

   g) Case:

      i) If *GTRC* simply contains <in paths clause> *IPC*, then let *NPV* be the number of <path variable>s simply contained in *IPC*, and let PVN_1, ... PVN_{NPV} be the <path variable>s simply contained in *IPC*. For all *p*, 1 (one) ≤ *p* ≤ *NPV*:

         1) Let IND_p be the 1-relative index of the <path pattern> that declares PVN_p among the list of <path pattern>s simply contained in *PPL*.

         2) Let CPB_p be the compressed path binding of the IND_p-th path binding of *M*.

         3) Let LEN_p be the number of ordered pairs in CPB_p.

ii) Otherwise, let *NPV* be the number of <path pattern>s simply contained in *PPL*. For all *p*, 1 (one) ≤ *p* ≤ *NPV*:

1) Let IND_p be *p*.

2) Let CPB_p be the compressed path binding of the *p*-th path binding of *M*.

3) Let LEN_p be the number of ordered pairs in CPB_p.

h) Let *T9* initially be an empty table.

i) Let *GTRC* be the <graph table rows clause> simply contained in *GT*. Let *GTCC* be the <graph table columns clause> simply contained in *GT*.

j) For each reduced match *M* in *SM*:

i) *M* is the *current match*, and the match number of *M* is the *current match number*.

ii) Case:

1) If *GTRC* is ONE ROW PER MATCH, then:

A) The General Rules of Subclause 9.11, "Applying bindings to generate a row", are applied with *GP* as *GRAPH PATTERN*, *GTRC* as *ROWS CLAUSE*, *GTCC* as *COLUMNS CLAUSE*, *MACH* as *MACHINERY*, *M* as *MULTI-PATH BINDING*, the null value as *PATH NUMBER*, and the null value as *OFFSET*; let *R* be the *ROW* returned from the application of those General Rules.

B) *R* is inserted into *T9*.

2) If *GTRC* is <one row per vertex>, then for each *p*, 1 (one)≤ *p* ≤ *NPV*, and for each odd integer *ODD* between 1 and LEN_p:

A) The General Rules of Subclause 9.11, "Applying bindings to generate a row", are applied with *GP* as *GRAPH PATTERN*, *GTRC* as *ROWS CLAUSE*, *GTCC* as *COLUMNS CLAUSE*, *MACH* as *MACHINERY*, *M* as *MULTI-PATH BINDING*, IND_p as *PATH NUMBER*, and *ODD* as *OFFSET*; let *R* be the *ROW* returned from the application of those General Rules.

B) *R* is inserted into *T9*.

3) If *GTRC* is <one row per step>, then for each *p*, 1 (one)≤ *p* ≤ *NPV*,

Case:

A) If LEN_p is 1 (one), then:

I) The General Rules of Subclause 9.11, "Applying bindings to generate a row", are applied with *GP* as *GRAPH PATTERN*, *GTRC* as *ROWS CLAUSE*, *GTCC* as *COLUMNS CLAUSE*, *MACH* as *MACHINERY*, *M* as *MULTI-PATH BINDING*, IND_p as *PATH NUMBER*, and 1 (one) as *OFF-SET*; let *R* be the *ROW* returned from the application of those General Rules.

II) *R* is inserted into *T9*.

B) Otherwise, for each odd integer *ODD* between 1 and LEN_p-2,

I) The General Rules of Subclause 9.11, "Applying bindings to generate a row", are applied with *GP* as *GRAPH PATTERN*, *GTRC* as *ROWS CLAUSE*, *GTCC* as *COLUMNS CLAUSE*, *MACH* as *MACHINERY*, *M* as

MULTI-PATH BINDING, IND_p as *PATH NUMBER*, and *ODD* as *OFFSET*; let *R* be the *ROW* returned from the application of those General Rules.

II) *R* is inserted into *T9*.

k) *T9* is the result of *TP*.

Conformance Rules

1) Insert after the last CR: Without Feature G900, “GRAPH_TABLE”, conforming SQL language shall not contain a <graph table derived table>.

2) Insert after the last CR: Without Feature G901, “GRAPH_TABLE: ONE ROW PER VERTEX”, conforming SQL language shall not contain <one row per vertex>.

3) Insert after the last CR: Without Feature G902, “GRAPH_TABLE: ONE ROW PER STEP”, conforming SQL language shall not contain <one row per step>.

4) Insert after the last CR: Without Feature G903, “GRAPH_TABLE: explicit ONE ROW PER MATCH keywords”, in conforming SQL language, <graph table rows clause> shall not specify ONE ROW PER MATCH.

5) Insert after the last CR: Without Feature G904, “All properties reference”, conforming SQL language shall not contain <all properties reference>.

6) Insert after the last CR: Without Feature G905, “GRAPH_TABLE: optional COLUMNS clause”, in conforming SQL language, <graph table shape> shall contain <graph table columns clause>.

7) Insert after the last CR: Without Feature G906, “GRAPH_TABLE: explicit EXPORT ALL”, in conforming SQL language,<graph table shape> shall not contain <graph table export clause> that simply contains EXPORT ALL SINGLETONS.

8) Insert after the last CR: Without Feature G907, “GRAPH_TABLE: EXPORT ALL EXCEPT”, conforming SQL language shall not contain an <export except>.

9) Insert after the last CR: Without Feature G908, “GRAPH_TABLE: EXPORT SINGLETONS list”, conforming SQL language shall not contain an <export variable list>.

10) Insert after the last CR: Without Feature G909, “GRAPH_TABLE: explicit EXPORT NO SINGLETONS”, conforming SQL language shall not contain EXPORT NO SINGLETONS.

11) Insert after the last CR: Without Feature G910, “GRAPH_TABLE: ‘in paths clause’”, conforming SQL language shall not contain an <in paths clause>.

7.2 <query specification>

This Subclause modifies Subclause 7.16, "<query specification>", in ISO/IEC 9075-2.

Function

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

Format

```
<select sublist> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <all properties reference>
```

NOTE 53 — <all properties reference> is transformed into a list of <column reference>s by syntactic transformations in Subclause 7.1, "<table reference>".

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after the last CR: Without Feature G904, "All properties reference", conforming SQL language shall not contain an <all properties reference>.

8 Predicates

This Clause modifies Clause 8, “Predicates”, in ISO/IEC 9075-2.

8.1 <predicate>

This Subclause modifies Subclause 8.1, “<predicate>”, in ISO/IEC 9075-2.

Function

Specify a condition that can be evaluated to give a Boolean value.

Format

```
<predicate> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <graph element predicate>

<graph element predicate> ::=
    <directed predicate>
  | <labeled predicate>
  | <source/destination predicate>
  | <all_different predicate>
  | <same predicate>
  | <bound predicate>
  | <property_exists predicate>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Augment GR 1) by adding “<graph element predicate>” to the list of immediately contained predicates.

2) Insert after GR 1): The result of <graph element predicate> is the result of the immediately contained <directed predicate>, <labeled predicate>, <source/destination predicate>, <all_different predicate>, <same predicate>, <bound predicate>, or <property_exists predicate>.

Conformance Rules

No additional Conformance Rules.

8.2 <directed predicate>

Function

Determine whether an edge variable is bound to a directed edge.

Format

```
<directed predicate> ::=
  <element reference> <directed predicate part 2>

<directed predicate part 2> ::=
  IS [ NOT ] DIRECTED
```

Syntax Rules

1) Let *ER* be the <element reference>. *ER* shall have singleton degree of reference, and shall reference an edge variable.

2) If NOT is specified, then the <directed predicate> is equivalent to:

   `NOT ( ER IS DIRECTED )`

Access Rules

None.

General Rules

1) Let *LOE* be the list of elements that are bound to *ER*.

2) The value of the <directed predicate>

   `ER IS DIRECTED`

   is determined as follows.

   Case:

   a) If *LOE* is empty, then *Unknown*.

   b) If the sole graph element in *LOE* is a directed edge, then *True*.

   c) Otherwise, *False*.

Conformance Rules

1) Without Feature G110, "IS DIRECTED predicate", conforming SQL language shall not contain a <directed predicate>.

8.3 <labeled predicate>

Function

Determine whether a graph element satisfies a <label expression>.

Format

```
<labeled predicate> ::=
  <element reference> <labeled predicate part 2>

<labeled predicate part 2> ::=
  IS [ NOT ] LABELED <label expression>
```

Syntax Rules

1) Let *ER* be the <element reference>. *ER* shall have singleton degree of reference.

2) Let *LE* be the <label expression>.

3) If NOT is specified, then the <labeled predicate> is equivalent to:

   `NOT ( ER IS LABELED LE )`

Access Rules

None.

General Rules

1) Let *LOE* be the list of elements that are bound to *ER*.

2) The value of the <labeled predicate> *LP*

   `ER IS LABELED LE`

   is determined as follows.

   Case:

   a) If *LOE* is empty, then the value of *LP* is <u>Unknown</u>.

   b) Otherwise, let *E* be the sole graph element in *LOE*. Let *LS* be the defined label set of *E*. The Syntax Rules of Subclause 9.4, “Satisfaction of a <label expression> by a defined label set”, are applied with *LE* as *LABEL EXPRESSION* and *LS* as *DEFINED LABEL SET*; let *TV* be the *TRUTH VALUE* returned from the application of those Syntax Rules. The value of *LP* is *TV*.

Conformance Rules

1) Without Feature G111, “IS LABELED predicate”, conforming SQL language shall not contain a <labeled predicate>.

8.4 <source/destination predicate>

Function

Determine whether a vertex is the source or destination of an edge.

Format

```
<source/destination predicate> ::=
    <vertex reference> <source predicate part 2>
  | <vertex reference> <destination predicate part 2>

<vertex reference> ::=
  <element reference>

<source predicate part 2> ::=
  IS [ NOT ] SOURCE OF <edge reference>

<destination predicate part 2> ::=
  IS [ NOT ] DESTINATION OF <edge reference>

<edge reference> ::=
  <element reference>
```

Syntax Rules

1) Let *VR* be the <vertex reference>. *VR* shall have singleton degree of reference, and shall reference a vertex variable.

2) Let *ER* be the <edge reference>. *ER* shall have singleton degree of reference, and shall reference an edge variable.

3) Let *SOD* be the <key word> SOURCE or DESTINATION simply contained in the <source/destination predicate>.

4) If NOT is specified, then the <source/destination predicate> is equivalent to:

```
NOT ( VR IS SOD OF ER )
```

Access Rules

None.

General Rules

1) Let *LOV* be the list of elements that are bound to *VR* and let *LOE* be the list of elements that are bound to *ER*.

2) The value of the <source/destination predicate> *SDP*

```
VR IS SOD OF ER
```

is determined as follows.

Case:

a) If *LOV* is empty or if *LOE* is empty, then the value of *SDP* is *Unknown*.

b) Otherwise, let *V* be the sole graph element in *LOV*, and let *E* be the sole graph element in *LOE*.

   Case:

   i) If *E* is an undirected edge, then the value of *SDP* is *False*.

   ii) If *SOD* is SOURCE and *V* is the source vertex of *E*, then the value of *SDP* is *True*.

   iii) If *SOD* is DESTINATION and *V* is the destination vertex of *E*, then the value of *SDP* is *True*.

   iv) Otherwise, the value of *SDP* is *False*.

Conformance Rules

1) Without Feature G112, "IS SOURCE and IS DESTINATION predicate", conforming SQL language shall not contain a <source/destination predicate>.

8.5 <all_different predicate>

Function

Determine whether all graph elements bound to a list of element references are pairwise different from one another.

Format

```
<all_different predicate> ::=
  ALL_DIFFERENT <left paren> <element reference> <comma> <element reference>
      [ { <comma> <element reference> }... ]
      <right paren>
```

Syntax Rules

1) Each <element reference> simply contained in <all_different predicate> *ADP* shall have unconditional singleton degree of reference.

Access Rules

None.

General Rules

1) Let *N* be the number of <element reference>s simply contained in *ADP*, and let ER_1, ..., ER_N be an enumeration of those <element reference>s.

2) For all *i*, 1 (one) ≤ *i* ≤ *N*, let LOE_i be the list of graph elements of ER_i, and let GE_i be the sole graph element in LOE_i.

3) The value of *ADP* is

   Case:

   a) If there exist *j*, *k* such that *j* < *k* and GE_j is the same graph element as GE_k, then *False*.

   b) Otherwise, *True*.

Conformance Rules

1) Without Feature G113, “ALL_DIFFERENT predicate”, conforming SQL language shall not contain an <all_different predicate>.

8.6 <same predicate>

Function

Determine whether all element references in a list of element references bind to the same graph element.

Format

```
<same predicate> ::=
  SAME <left paren> <element reference> <comma> <element reference>
      [ { <comma> <element reference> }... ]
      <right paren>
```

Syntax Rules

1) Each <element reference> simply contained in <same predicate> *SP* shall have unconditional singleton degree of reference.

Access Rules

None.

General Rules

1) Let *N* be the number of <element reference>s simply contained in *SP*, and let ER_1, ..., ER_N be an enumeration of those <element reference>s.

2) For all *i*, 1 (one) ≤ *i* ≤ *N*, let LOE_i be the list of graph elements of ER_i and let GE_i be the sole graph element in LOE_i.

3) The value of *SP* is

   Case:

   a) If every GE_i, 1 (one) ≤ *i* ≤ *N*, is the same graph element, then *True*.

   b) Otherwise, *False*.

Conformance Rules

1) Without Feature G114, "SAME predicate", conforming SQL language shall not contain a <same predicate>.

8.7 <bound predicate>

Function

Determine whether an element variable has any bindings.

Format

```
<bound predicate> ::=
  <element reference> <bound predicate part 2>

<bound predicate part 2> ::=
  IS [ NOT ] BOUND [ AS <primary element variable> ]

<primary element variable> ::=
  <element variable>
```

Syntax Rules

1) Let *ER* be the <element reference>.

2) Case:

   a) If <primary element variable> *PEV* is specified, then:

      i) The <bound predicate> shall be contained in the <graph table columns clause> of a <graph table> *GT*, or in the scope of the exported <graph pattern variable>s of a <graph table> *GT*.

      ii) *GT* shall specify <one row per iteration> *ORPI*.

      iii) *ER* shall reference an iterator variable declared by *ORPI*.

      iv) The <element variable> simply contained in *PEV* shall identify a primary element variable.

      v) Let *ASPEV* be:

      ```
      AS PEV
      ```

   b) Otherwise, let *ASPEV* be the zero-length string.

3) If NOT is specified, then the <bound predicate> is equivalent to:

   ```
   NOT ( ER IS BOUND ASPEV )
   ```

Access Rules

None.

General Rules

1) Let *LOE* be the list of elements that are bound to *ER*.

2) If <primary element variable> *PEV* is specified, then let *LOPV* be the current list of primary variable names of *ER*.

NOTE 54 — This list is specified in the General Rules of Subclause 9.11, “Applying bindings to generate a row”.

3) The value of the <bound predicate>

```
ER IS BOUND ASPEV
```

is determined as follows.

Case:

a) If *LOE* is empty, then *False*.

b) If <primary element variable> *PEV* is specified and *PEV* is not equivalent to a member of *LOPV*, then *False*.

c) Otherwise, *True*.

Conformance Rules

1) Without Feature G810, “IS BOUND predicate”, conforming SQL language shall not contain a <bound predicate>.

2) Without Feature G811, “IS BOUND predicate: AS option”, <bound predicate> shall not contain <primary element variable>.

8.8 <property_exists predicate>

Function

Determine if the graph element bound to a singleton element reference has a property.

Format

```
<property_exists predicate> ::=
  PROPERTY_EXISTS <left paren> <element reference> <comma> <property name> <right paren>
```

Syntax Rules

1) Let *ER* be the <element reference> and let *PN* be the <property name>.

2) *ER* shall have singleton degree of reference.

Access Rules

None.

General Rules

1) Let *LOE* be the list of graph elements that are bound to *ER*.

2) The value of the <property_exists predicate> is

   Case:

   a) If *LOE* is empty, then *Unknown*.

   b) If the sole graph element of *LOE* has property *PN*, then *True*.

   c) Otherwise, *False*.

Conformance Rules

1) Without Feature G115, “PROPERTY_EXISTS predicate”, conforming SQL language shall not contain <property_exists predicate>.

9 Additional common rules

This Clause modifies Clause 9, "Additional common rules", in ISO/IEC 9075-2.

9.1 Potential sources of non-determinism

This Subclause modifies Subclause 9.16, "Potential sources of non-determinism", in ISO/IEC 9075-2.

Function

Define the potential sources of non-determinism.

Syntax Rules

1) Insert after SR 1)as): An <any path search>, an <any shortest path search> or a <counted shortest path search>.

2) Insert after SR 1)as): A <graphical match number function>.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

9.2 Contextual inference of a set of labels

Function

Infer the set of labels of graph elements that might bind to an element variable.

Subclause Signature

```
"Contextual inference of a set of labels" [Syntax Rules] (
  Parameter: "SQL-PROPERTY GRAPH",
  Parameter: "GRAPH PATTERN",
  Parameter: "ELEMENT VARIABLE",
  Parameter: "CONTEXT"
) Returns: "SET OF LABELS"
```

SQL-PROPERTY GRAPH — an SQL-property graph.

GRAPH PATTERN — a <graph pattern>.

ELEMENT VARIABLE — an element variable declared by GRAPH PATTERN.

CONTEXT — reference to an instance of a BNF non-terminal, providing context for implementation-defined inferencing of the labels of graph elements that might be bound to the element variable in that context.

SET OF LABELS — the set of labels that might bind to ELEMENT VARIABLE.

Syntax Rules

1) Let *SPG* be the *SQL-PROPERTY GRAPH*, let *GP* be the *GRAPH PATTERN*, let *V* be the *ELEMENT VARIABLE*, and let *CON* be the *CONTEXT* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is returned as *SET OF LABELS*.

2) Let *ILE*(*P*) be a partial function of one argument, *P*, which is a <path pattern list> or <path pattern expression>, whose value (when defined) is a <label expression>, defined recursively as follows.

   Case:

   a) If *P* is an <element pattern>, then

      Case:

      i) If *P* does not declare *V*, then *ILE*(*P*) is undefined.

      ii) If *P* does not have a <label expression>, then *ILE*(*P*) is %|!%.

      iii) Otherwise, *ILE*(*P*) is the <label expression> simply contained in *P*.

   b) If *P* is a <path concatenation>, then let *Q* be the <path term> and let *R* be the <path factor> simply contained in *P*.

      Case:

      i) If *ILE*(*Q*) and *ILE*(*R*) are both defined, then *ILE*(*P*) is (*ILE*(*Q*)) & (*ILE*(*R*))

      ii) If *ILE*(*Q*) is defined, then *ILE*(*P*) is *ILE*(*Q*).

      iii) If *ILE*(*R*) is defined, then *ILE*(*P*) is *ILE*(*R*).

iv) Otherwise, *ILE*(*P*) is undefined.

c) If *P* is a <quantified path primary> or a <questioned path primary>, then let *Q* be the <path primary> simply contained in *P*.

Case:

i) If *ILE*(*Q*) is defined, then *ILE*(*P*) is *ILE*(*Q*).

ii) Otherwise, *ILE*(*P*) is undefined.

d) If *P* is a <path pattern union> or a <path multiset alternation>, then let Q_1, ..., Q_n be the <path term>s simply contained in *P* such that *ILE*(Q_i) is defined.

Case:

i) If $n > 0$, then *ILE*(*P*) is (*ILE*(Q_1)) | ... | (*ILE*(Q_n))

ii) Otherwise, *ILE*(*P*) is undefined.

e) If *P* is a <path pattern list>, then let Q_1, ..., Q_n be the <path pattern>s simply contained in *P* such that *ILE*(Q_i) is defined; then

Case:

i) If $n > 0$, then *ILE*(*P*) is (*ILE*(Q_1)) & ... & (*ILE*(Q_n))

ii) Otherwise, *ILE*(*P*) is undefined.

f) If *P* and *Q* are instances of BNF non-terminals such that *P* ::= *Q*, then *ILE*(*P*) is *ILE*(*Q*).

3) Case:

a) If there is an implementation-defined (IE012) rule that correctly deduces from the defined label sets and edge triplets of *SPG*, from *GP*, and from the context *CON* a <label expression> that must be satisfied by every binding to *V* in the context *CON*, then let *IDLE* be that <label expression>.

NOTE 55 — For example, given the <graph pattern>:

```
(P) -> (   (P IS L1 WHERE P.X = 1)
         | (P IS L2 WHERE P.Y = 2) )
```

the mandatory rules of this Subclause infer that P satisfies the <label expression> L1 | L2. However, in the context of the <element pattern> (P IS L1 WHERE P.X = 1), an SQL-implementation might infer the tighter <label expression> L1.

b) Otherwise, let *IDLE* be %|!%.

4) Let *LE* be:

```
( ILE(GP) ) & ( IDLE )
```

NOTE 56 — Although *ILE* is a partial function, it is defined for any <path pattern list> or <path pattern expression> that declares *V*. This Subclause assumes that *GP* declares *V*.

5) A defined label set *LS* is *satisfactory for V* if all of the following true:

a) Case:

i) If *V* is a vertex variable, then *LS* is a vertex defined label set.

ii) If *V* is an edge variable that is declared only in <full edge pointing left>, <full edge pointing right>, or <full edge left or right>, then *LS* is an edge defined label set for edges that are always directed or edges that are possibly directed or undirected.

iii) If *V* is an edge variable that is declared only in <full edge undirected>, then *LS* is an edge defined label set for edges that are always undirected or edges that are possibly directed or undirected.

iv) Otherwise, *LS* is an edge defined label set.

b) *True* is the *TRUTH VALUE* returned as *TV* when the Syntax Rules of Subclause 9.4, "Satisfaction of a <label expression> by a defined label set", are applied with *LE* as *LABEL EXPRESSION* and *LS* as *DEFINED LABEL SET*.

6) Let *SOLS* be the set of defined label sets that are satisfactory for *V*.

7) Let *SOL* be the set union of the defined label sets comprising *SOLS*.

NOTE 57 — *SOL* is not necessarily a defined label set. For example, a graph can have vertex defined label sets {Person, Employee} and {Person, Student} but no defined label set having both Employee and Student. Given the <vertex pattern>

```
(P IS Person)
```

and assuming no other inferencing than implied by this <vertex pattern>, then *SOL* is {Person, Employee} ∪ {Person, Student} = {Person, Employee, Student}, which is not a defined label set. This implies that the properties of P are the union of the properties of the labels Person, Employee, and Student.

8) Evaluation of the Syntax Rules is terminated and control is returned to the invoking Subclause, which receives *SOL* as *SET OF LABELS*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.3 Expansion of an <all properties reference>

Function

Specify the syntactic transformation to expand an <all properties reference>.

Subclause Signature

```
“Expansion of an <all properties reference>” [Syntax Rules] (
  Parameter: “SQL-PROPERTY GRAPH”,
  Parameter: “GRAPH PATTERN”,
  Parameter: “ELEMENT VARIABLE”
) Returns: “LIST OF PROPERTY REFERENCES”
```

SQL-PROPERTY GRAPH — an SQL-property graph.

GRAPH PATTERN — a <graph pattern>.

ELEMENT VARIABLE — an element variable declared by GRAPH PATTERN.

LIST OF PROPERTY REFERENCES — the list of the names of the properties of ELEMENT VARIABLE.

Syntax Rules

1) Let *SPG* be the *SQL-PROPERTY GRAPH*, let *GP* be the *GRAPH PATTERN*, and let *EV* be the *ELEMENT VARIABLE* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is returned as *LIST OF PROPERTY REFERENCES*.

2) The Syntax Rules of Subclause 9.2, “Contextual inference of a set of labels”, are applied with *SPG* as *SQL-PROPERTY GRAPH*, *GP* as *GRAPH PATTERN*, *EV* as *ELEMENT VARIABLE*, and *GP* as *CONTEXT*; let *SOL* be the *SET OF LABELS* returned from the application of those Syntax Rules.

3) Let *SOPN* be the set of property names of the properties of the labels of *SOL*.

4) Let P_1, ..., P_n be an enumeration of the names of the properties of *SOPN*, in an implementation-dependent (US017) order.

5) Case:

   a) If $n = 0$ (zero), then let *LOPR* be a zero-length character string.

   b) Otherwise, let *LOPR* be:

      $EV.P_1$, ..., $EV.P_n$

6) Evaluation of the Syntax Rules is terminated and control is returned to the invoking Subclause, which receives *LOPR* as *LIST OF PROPERTY REFERENCES*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.4 Satisfaction of a <label expression> by a defined label set

Function

Determine if a defined label set satisfies a <label expression>.

Subclause Signature

```
"Satisfaction of a <label expression> by a defined label set" [Syntax Rules] (
  Parameter: "LABEL EXPRESSION",
  Parameter: "DEFINED LABEL SET"
) Returns: "TRUTH VALUE"
```

LABEL EXPRESSION — a <label expression>.

DEFINED LABEL SET — the defined label set of a graph element.

TRUTH VALUE — *True* if the DEFINED LABEL SET satisfies the LABEL EXPRESSION; otherwise *False*.

Syntax Rules

1) Let *LEXP* be the *LABEL EXPRESSION* and let *LS* be the *DEFINED LABEL SET* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is returned as *TRUTH VALUE*.

2) A defined label set *LS satisfies* a <label expression> *LE* according to the following recursive definition:

   a) If *LE* is a <label name> *L2*, then *L2* is a member of *LS*.

   b) If *LE* is a <wildcard label>, then *LS* is non-empty.

      NOTE 58 — This condition is always true; every defined label set is non-empty. The rule is written this way in case empty defined label sets are permitted in the future, or for guidance to an SQL-implementation that supports graph elements with no labels.

   c) If *LE* is a <parenthesized label expression> *PLE*, then let *LE2* be the <label expression> simply contained in *PLE*; *LS* satisfies *LE2*.

   d) If *LE* is a <label negation>, then let *LP* be the <label primary> simply contained in *LE*; *LS* does not satisfy *LP*.

   e) If *LE* is a <label conjunction>, then let *L1* be the <label term> and let *L2* be the <label factor> simply contained in *LE*; *LS* satisfies *L1* and *LS* satisfies *L2*.

   f) If *LE* is a <label disjunction>, then let *L1* be the <label expression> and let *L2* be the <label term> simply contained in *LE*; *LS* satisfies *L1* or *LS* satisfies *L2*.

3) *TV* is

   Case:

   a) If *LS* satisfies *LEXP*, then *True*.

   b) Otherwise, *False*.

4) Evaluation of the Syntax Rules is terminated and control is returned to the invoking Subclause, which receives *TV* as *TRUTH VALUE*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.5 Converting a tabular property graph to a pure property graph

Function

Convert a property graph represented in tables to a pure property graph.

Subclause Signature

```
“Converting a tabular property graph to a pure property graph” [General Rules] (
  Parameter: “SQL-PROPERTY GRAPH”
) Returns: “PURE PROPERTY GRAPH”
```

SQL-PROPERTY GRAPH — an SQL-property graph.

PURE PROPERTY GRAPH — the pure property graph derived from the tabular property graph descriptor included in SQL-PROPERTY GRAPH’s descriptor.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *SPG* be the *SQL-PROPERTY GRAPH* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *PURE PROPERTY GRAPH*.

2) Except where explicitly specified, the General Rules of this Subclause are not terminated if an exception condition is raised.

3) Let *SPGD* be the SQL-property graph descriptor of *SPG*.

4) Let *PPGD* be the pure property graph descriptor included in *SPGD*.

5) Let *TPGD* be the tabular property graph descriptor included in *SPGD*.

6) The tabular property graph described by *TPGD* is converted to a pure property graph *PPG*, which conforms to the graph schema described by *PPGD*, as follows:

   a) For each vertex table whose descriptor *VTD* is included in *TPGD* and each row in that table, a vertex *V* is created and included in *PPG* as follows:

      i) *V* is assigned an implementation-dependent (UV021) identifier that is unique within *PPG*.

      ii) For each label *L* included in *VTD*, *V* is assigned a label with the same identifier as *L*. All labels assigned to *V* constitute the *label set* of *V*.

      iii) For each unique property included in *VTD*, *V* has a property with the same name and declared type. For each property that is shared between labels, *V* has exactly one property with the same name and declared type. The value of the property *P* of *V* is obtained by evaluating the <value expression> *VE* associated with the property included in *VTD*. If during the evaluation of *VE* an exception condition *EC* is raised, then *P* is flagged as *unusable* with *EC* as additional information.

NOTE 59 — In the case of a property shared between labels, it is immaterial which one is used to determine the name, declared type, and <value expression> of the property of *V* as these attributes of the property are all identical among the labels sharing the property.

b) For each edge table whose descriptor *ETD* is included in *TPGD* and each row *ER* in that table:

i) Let *n* be the number of vertices that were created in GR 6)a) from a row *SR* in the source vertex table included in *ETD* such that the value of the edge source key of *ER* is equal to the value of the source vertex key of *SR*. Let SV_i, 1 (one) ≤ *i* ≤ *n*, be an enumeration of those vertices.

ii) Let *m* be the number of vertices that were created in GR 6)a) from a row *DR* in the destination vertex table included in *ETD* such that the value of the edge destination key of *ER* is equal to the value of the destination vertex key of *DR*. Let DV_j, 1 (one) ≤ *j* ≤ *m*, be an enumeration of those vertices.

iii) Case:

1) If *n* > 1 (one) or m > 1 (one), and the SQL-implementation does not support Feature G940, “Multi-sourced/destined edges”, then an exception condition is raised: *data exception — multi-sourced or multi-destined edge (22G0K)* and no further General Rules of this Subclause are applied.

2) If *n* = 0 (zero) or *m* = 0 (zero), and the SQL-implementation does not support Feature G941, “Implicit removal of incomplete edges”, then an exception condition is raised: *data exception — incomplete edge (22G0L)* and no further General Rules of this Subclause are applied.

3) Otherwise, for *i*, 1 (one) ≤ *i* ≤ *n* and *j*, 1 (one) ≤ *j* ≤ *m*:

A) A directed edge *E* is created.

NOTE 60 — If *n* is 1 (one) and *m* is 1 (one), then exactly 1 (one) edge is created. Otherwise, *n***m* edges are created, which in turn means that if *n* or *m* is 0 (zero), then no edge is created.

B) SV_i is the source vertex of *E*.

C) DV_j is the destination vertex of *E*.

D) *E* is assigned an implementation-dependent (UV022) identifier that is unique within *PPG*.

E) For each label *L* included in *ETD*, *E* is assigned a label with the same identifier as *L*. All labels assigned to *E* constitute the *label set* of *E*.

F) For each unique property included in *ETD*, *E* has a property with the same name and declared type. For each property that is shared between labels, *E* has exactly one property with the same name and declared type. The value of the property *P* of *E* is obtained by evaluating the <value expression> *VE* associated with the property included in *ETD*. If during the evaluation of *VE* an exception condition *EC* is raised, then *P* is flagged as *unusable* with *EC* as additional information.

NOTE 61 — In the case of a property shared between labels, it is immaterial which one is used to determine the name, declared type, and <value expression> of the property of *E* as these attributes of the property are all identical among the labels sharing the property.

7) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *PPG* as *PURE PROPERTY GRAPH*.

Conformance Rules

None.

9.6 Machinery for graph pattern matching

Function

Define the infrastructure (alphabet, mappings and related definitions) used in graph pattern matching.

Subclause Signature

```
"Machinery for graph pattern matching" [General Rules] (
  Parameter: "PURE PROPERTY GRAPH",
  Parameter: "PATH PATTERN LIST"
) Returns: "MACHINERY"
```

PURE PROPERTY GRAPH — a pure property graph.

PATH PATTERN LIST — a <path pattern list>.

MACHINERY — the machinery to be used for graph pattern matching.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *PG* be the *PURE PROPERTY GRAPH* and let *PPL* be the *PATH PATTERN LIST* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *MACHINERY*.

2) Let *SVV* be the set of names of vertex variables declared in *PPL* at the same depth of graph pattern matching, and let *SEV* be the set of names of edge variables declared in *PPL* at the same depth of graph pattern matching.

3) For each subpath variable *SPV* declared in *PPL* at the same depth of graph pattern matching, let *SPVBEGIN* and *SPVEND* be two distinct <identifier>s that are distinct from every <identifier> in *SVV* ∪ *SEV* and from every <identifier> created by this rule. *SPVBEGIN* is the *begin subpath symbol* and *SPVEND* is the *end subpath symbol* associated with *SPV*. Let *SPS* be the set of every subpath symbol.

4) Let *"()"* and *"-"* be mutually distinct <identifier>s that are distinct from every <identifier> in *SVV* ∪ *SEV* ∪ *SPS*. These are, respectively, the *anonymous vertex symbol* and the *anonymous edge symbol*. Let *SAS* be the set of anonymous symbols.

5) Let *NPP* be the number of <parenthesized path pattern expression>s contained in *PPL* at the same depth of graph pattern matching. Let $PPPE_1$, ..., $PPPE_{NPP}$ be an enumeration of the <parenthesized path pattern expression>s contained in *PPL* at the same depth of graph pattern matching. For each *i*, 1 (one) ≤ *i* ≤ *NPP*, *i* is the *bracket index* of $PPPE_i$.

6) Let "$[_1$", ..., "$[_{NPP}$", "$]_1$", ..., "$]_{NPP}$" be 2 * *NPP* <identifier>s that are mutually distinct, and distinct from every member of *SVV* ∪ *SEV* ∪ *SPS* ∪ *SAS* and from every graph element of *PG*. These are called *bracket*

symbols. For every bracket symbols "$[_j$" or "$]_j$", *j* is the *bracket index* of the bracket symbol. There are two bracket symbols for each bracket index *j* between 1 (one) and *NPP*, corresponding to the <parenthesized path pattern expression>s $PPPE_j$. Let *SBS* be the set of bracket symbols.

7) Let *GX* be the set whose members are the graph elements of *PG*, the subpath symbols, and the bracket symbols.

8) Let *ABC* be *SPS* ∪ *SBS* ∪ *SVV* ∪ *SEV* ∪ *SAS*. *ABC* is the *alphabet*. The members of *ABC* are *symbols*.

9) A *word* is a string of elements of *ABC*.

10) An *elementary binding* is a pair (*LET*, *GE*) where *LET* is a member of *ABC* and *GE* is a member of *GX*, such that:

    Case:

    a) If *LET* is a bracket symbol, then *LET* = *GE*. In this case, the elementary binding is a *bracket symbol binding*. If *LET* is a start bracket symbol, then the elementary binding is a *start bracket symbol binding*; otherwise, it is an *end bracket symbol binding*. The bracket index of *LET* is the bracket index of the bracket symbol binding.

    b) If *LET* is a subpath symbol, then *LET* = *GE*. In this case, the elementary binding is a *subpath symbol binding*.

    c) If *LET* is the name of a vertex variable or the anonymous vertex symbol, then *GE* is a vertex. In this case the elementary binding is a *vertex symbol binding*.

    d) If *LET* is the name of an edge variable or the anonymous edge symbol, then *GE* is an edge. In this case, the elementary binding is an *edge symbol binding*.

11) If *EB* = (*LET*, *GE*) is an elementary binding, then *EB* is an *elementary binding of LET*, and *EB binds LET* to *GE*.

    NOTE 62 — An elementary binding is a mapping of a symbol (an <identifier>) to a member of *GX*; it is not the binding of a graph pattern variable. In particular, if *LET* is the name of an element variable *EV*, there can be more than one elementary binding of *LET* in a multi-path binding. In a consistent path binding (defined subsequently), two elementary bindings of *LET* necessarily bind to the same graph element in contexts in which *LET* is exposed as unconditional singleton; otherwise elementary bindings of *LET* are independent of one another, and can bind to more than one graph element. Similarly, references to *EV* are context-dependent, and can resolve to a list that is a proper subset of all the graph elements bound to *LET* by elementary bindings. Resolution of <element reference>s is performed by the General Rules of Subclause 9.9, “Applying bindings to evaluate an expression”.

12) A *compressed binding* is a pair (*LOV*, *GE*) where *LOV* is a list of names of primary variables and *GE* is a graph element.

13) A *path binding* is a sequence of zero or more elementary bindings, $B = (LET_1, E_1), ..., (LET_N, E_N)$. Given a path binding *B*:

    NOTE 63 — Unlike the definition of path, the definition of path binding also allows a sequence of zero elementary bindings. For example, a quantifier can iterate 0 (zero) times, resulting in an empty path binding. The result of a <path pattern>, on the other hand, is unable to be empty because of a Syntax Rule that enforces a minimum vertex count of 1 (one).

    a) The *word* of *B* is the sequence $LET_1, ..., LET_N$.

    b) The *annotated path* of *B* is the sequence $E_1, ..., E_N$.

        NOTE 64 — If the path binding is consistent (defined subsequently), then the annotated path of *B* contains within it a path that matches the word of *B*, plus mark-up with bracket symbols and subpath symbols indicating how to interpret the path as a match to the word.

    c) The *compressed path binding CPB* of *B* is obtained from *B* as follows:

        i) Let *CPB* be a copy of *B*.

ii) All subpath symbol bindings and all bracket symbol bindings are deleted from *CPB*.

iii) Each maximal subsequence *MS* of *CPB* comprising one or more consecutive vertex bindings is replaced by a single compressed binding, whose components are the following:

1) The first component is a list of the names of vertex variables in the first component of the vertex bindings of *MS*.

NOTE 65 — This list is empty if only the anonymous vertex symbol is bound in *MS*.

2) The second component is the vertex that is bound by the first vertex binding in *MS*.

NOTE 66 — For consistent path bindings, consecutive vertex bindings will bind the same vertex.

iv) In each edge binding *EB* of *CPB*, the first component is replaced by a list of zero or one name of an edge variable, retaining the name of the edge variable in the first component of *EB*, if any.

d) The *extracted path XP* of *B* is obtained from the compressed path binding of *B* as the sequence of the second components of the compressed bindings of *CPB*.

NOTE 67 — The extracted path is a path if the path binding is consistent and non-empty; otherwise, the extracted path is not necessarily a path.

14) Let *REDUCE* be a function that maps path bindings to path bindings, defined as follows.

a) Let *PBIN* be a path binding.

b) Let *PBOUT* be a copy of *PBIN*.

c) All bracket bindings are removed from *PBOUT*.

d) The following steps are performed on *PBOUT* repeatedly until no more anonymous vertex bindings can be removed:

i) If there are two adjacent anonymous vertex bindings, then the second is removed.

ii) If there is a binding of a vertex variable adjacent to an anonymous vertex binding, then the anonymous vertex binding is removed.

e) *REDUCE*(*PBIN*) is *PBOUT*.

15) In a path binding, two vertex bindings are *separable* if there is an edge binding between them.

16) A path binding *B* is *consistent* if all of the following are true:

a) The extracted path *XP* of *B* is either the empty sequence or a path of *PG*.

NOTE 68 — That is, either *XP* is empty or *XP* begins with a vertex, alternates between vertices and edges, each edge in *XP* connects the vertex before and after it, and *XP* ends with a vertex.

b) For every two vertex bindings (LET_1, E_1) and (LET_2, E_2) that are not separable, $E_1 = E_2$.

NOTE 69 — If there are only bracket symbol or subpath symbol bindings between two vertex bindings, then the vertex bindings must bind to the same vertex.

c) For every edge binding *EB* = (*LET*, *GE*), let (LET_{left}, GE_{left}) be the last vertex binding to the left of *EB* and let $(LET_{right}, GE_{right})$ be the first vertex binding to the right of *EB*.

Case:

i) If *GE* is an undirected edge, then *GE* is an edge connecting GE_{left} and GE_{right}.

ii) If *GE* is a directed edge, then either GE_{left} is the source vertex and GE_{right} is the destination vertex of *GE*, or GE_{right} is the source vertex and GE_{left} is the destination vertex of *GE*.

NOTE 70 — The directionality constraint of the edge binding is fully checked during the generation of the regular language for <path concatenation>.

d) For every start bracket symbol binding *SBS* contained in *B*, let *j* be the bracket index of *SBS*. All of the following are true:

i) There is an end bracket symbol binding contained in *B* and following *SBS* whose bracket index is *j*. Let *EBS* be the first such end bracket symbol binding. Let *PPS* be the explicit or implicit <path mode prefix> simply contained in the <parenthesized path pattern expression> whose bracket index is *j*. Let *PM* be the <path mode> contained in *PPS*.

ii) Case:

1) If *PM* is TRAIL, then there is no pair of edge bindings at two different positions between *SBS* and *EBS* that bind the same edge.

NOTE 71 — This definition does not take note of the symbols that are bound, only the edges. It is a violation of the TRAIL <path mode> if the symbols in the pair of distinct edge bindings are both anonymous edge symbols, are both edge variables (whether the same or different), or one is the anonymous edge symbol and the other is an edge variable.

2) If *PM* is SIMPLE, then no two separable vertex bindings between *SBS* and *EBS* bind the same vertex, except that the first and last vertex binding between *SBS* and *SES* may bind the same vertex.

NOTE 72 — This definition does not take note of the symbols that are bound, only the vertices. It is a violation of the SIMPLE <path mode> if the symbols in the pair of separable vertex bindings are both anonymous vertex symbols, are both vertex variables (whether the same or different), or one is the anonymous vertex symbol and the other is a vertex variable. However, the first and last vertex binding between *SBS* and *EBS* can bind the same vertex without violating the SIMPLE <path mode>.

3) If *PM* is ACYCLIC, then no two separable vertex bindings between *SBS* and *EBS* bind the same vertex.

NOTE 73 — This definition does not take note of the symbols that are bound, only the vertices. It is a violation of the ACYCLIC <path mode> if the symbols in the pair of separable vertex bindings are both anonymous vertex symbols, both are vertex variables (whether the same or different), or one is the anonymous vertex symbol and the other is a vertex variable.

NOTE 74 — The <path mode> WALK imposes no constraints on the extracted path.

17) A *multi-path binding* is an n-tuple (PB_1, ..., PB_n) for some positive integer *n* such that each PB_i is a path binding.

18) *REDUCE* is extended to multi-path bindings as follows. If *MPB* = (PB_1, ..., PB_n) is a multi-path binding, then *REDUCE*(*MPB*) = (*REDUCE*(PB_1), ..., *REDUCE*(PB_n)).

19) If *S* and *T* are sets of strings, then let $S \cdot T$ be the set of strings formed by concatenating an element of *S* followed by an element of *T*, that is, $S \cdot T = \{ s\, t \mid s$ is an element of S, t is an element of $T \}$.

NOTE 75 — The · operator will be used to concatenate words (strings of symbols) and path bindings (strings of elementary bindings).

20) If *S* is a set of strings, then let S^0 be the set whose only element is the string of length 0 (zero), and for each non-negative integer *n*, let S^{n+1} be $S^n \cdot S$. Let S^* be the union of S^n for every non-negative integer *n*.

NOTE 76 — If S is not empty, then S^* is an infinite set; however, a finite result for every <graph pattern> is assured by the syntactic requirement that every <quantified path primary> is bounded, contained in a restrictive <parenthesized path pattern expression> or contained in a selective <path pattern>.

21) Let *MACH* be a data structure comprising the following:

   a) *ABC*, the alphabet, formed as the disjoint union of the following:

      i) *SVV*, the set of names of vertex variables.

      ii) *SEV*, the set of names of edge variables.

      iii) *SPS*, the set of subpath symbols.

      iv) *SAS*, the set of anonymous symbols.

      v) *SBS*, the set of bracket symbols.

   b) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.

22) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *MACH* as *MACHINERY*.

Conformance Rules

None.

9.7 Evaluation of a <path pattern expression>

Function

Evaluate a <path pattern expression>.

Subclause Signature

```
"Evaluation of a <path pattern expression>" [General Rules] (
  Parameter: "PURE PROPERTY GRAPH",
  Parameter: "PATH PATTERN LIST",
  Parameter: "MACHINERY",
  Parameter: "SPECIFIC BNF INSTANCE"
) Returns: "SET OF MATCHES"
```

PURE PROPERTY GRAPH — a pure property graph.

PATH PATTERN LIST — a <path pattern list>.

MACHINERY — the machinery for graph pattern matching.

SPECIFIC BNF INSTANCE — the specific instance of a BNF non-terminal to be evaluated.

SET OF MATCHES — the set of local matches to the SPECIFIC BNF INSTANCE.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *PG* be the *PURE PROPERTY GRAPH*, let *PPL* be the *PATH PATTERN LIST*, let *MACH* be the *MACHINERY*, and let *SBI* be the *SPECIFIC BNF INSTANCE* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *SET OF MATCHES*.

2) The following components of *MACH* are identified:

   a) *ABC*, the alphabet, formed as the disjoint union of the following:

      i) *SVV*, the set of names of vertex variables.

      ii) *SEV*, the set of names of edge variables.

      iii) *SPS*, the set of subpath symbols.

      iv) *SAS*, the set of anonymous symbols.

      v) *SBS*, the set of bracket symbols.

   b) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.

3) For every instance *BNT* of a BNF non-terminal that is a <path pattern expression>, <path term>, <path pattern union>, <path factor>, <path concatenation>, <path primary>, <quantified path primary>, <questioned path primary>, <element pattern>, or <parenthesized path pattern expression> equal to or contained in *SBI* at the same depth of graph pattern matching, the following are defined by simultaneous recursion: the *regular language of BNT*, denoted *RL*(*BNT*), defined as a set of words over *ABC*; and the *set of local matches to BNT*, denoted *SLM*(*BNT*), defined as a set of path bindings.

NOTE 77 — *SLM*(*BNT*) is consistent except when concatenating an edge pattern prior to concatenating the following vertex pattern. Restrictive path modes are enforced when generating *SLM*(*BNT*) for the <parenthesized path pattern expression> that declares the path mode.

NOTE 78 — *RL*(*BNT*) and *SLM*(*BNT*) can be infinite sets if *BNT* contains an effectively unbounded quantifier. Every effectively unbounded quantifier is required to be contained in a selective <path pattern>; the potentially infinite set of local matches is subsequently reduced to a finite set by the General Rules of Subclause 9.8, "Evaluation of a selective <path pattern>".

NOTE 79 — The BNF non-terminals are listed above in the "top down" order of appearance in the Format of Subclause 10.6, "<path pattern expression>"; the definitions in the following subrules treat the same BNF non-terminals in "bottom up" order.

Case:

a) If *BNT* is a <parenthesized path pattern expression>, then let *PPE* be the <path pattern expression> immediately contained in *BNT* and let *j* be the bracket index of *BNT*.

NOTE 80 — "Bracket index" is defined in Subclause 9.6, "Machinery for graph pattern matching".

i) Case:

1) If *BNT* simply contains a <subpath variable declaration> *SVD*, then let *SV* be the subpath variable declared by *SVD*. Let *BSV* be the begin subpath symbol associated with *SV* and let *ESV* be the end subpath symbol associated with *SV*. Let *BSVBINDING* be (*BSV*, *BSV*) and let *ESVBINDING* be (*ESV*, *ESV*).

2) Otherwise, let *BSV*, *ESV*, *BSVBINDING* and *ESVBINDING* be the zero-length string.

ii) *RL*(*BNT*) is $\{ "[_j" \} \cdot \{ BSV \} \cdot RL(PPE) \cdot \{ ESV \} \cdot \{ "]_j" \}$

iii) Let *STPB* be *SLM*(*PPE*).

Let *SLMMAYBE* be $\{ ("[_j", "[_j") \} \cdot \{ BSVBINDING \} \cdot STPB \cdot \{ ESVBINDING \} \cdot \{ ("]_j", "]_j") \}$

SLM(*BNT*) is the set of every path binding in *SLMMAYBE* that is consistent.

NOTE 81 — That is, the words in *RL*(*BNT*) are formed by surrounding the words of *RL*(*PPE*) by the bracket symbols "$[_j$" and "$]_j$". If *BNT* contains a subpath variable declaration, then the words of *RL*(*BNT*) are also surrounded by the begin and end subpath symbol associated with that subpath variable. Similarly, the path bindings in *SLMMAYBE* are formed by surrounding the path bindings with bracket bindings and, if there is a subpath declaration, with the corresponding begin and end subpath bindings. Eliminating inconsistent bindings from *SLMMAYBE* to get *SLM*(*BNT*) has the effect of enforcing restrictive path modes.

b) If *BNT* is an <element pattern> *EP*, then

i) Case:

1) If *EP* declares an element variable *EV*, then let *EPI* be the name of *EV*.

2) If *EP* is a <vertex pattern>, then let *EPI* be "()", the anonymous vertex symbol.

3) If *EP* is an <edge pattern>, then let *EPI* be "-", the anonymous edge symbol.

ii) *RL*(*BNT*) is { *EPI* }, the set whose sole member is *EPI*.

iii) *SLM*(*BNT*) is the set of every elementary binding (*EPI*, *GE*) such that:

1) If *EP* is a <vertex pattern>, then *GE* is a vertex.

2) If *EP* is an <edge pattern>, then *GE* is an edge.

3) If *EP* simply contains a <label expression> *LE*, then _*True*_ is the *TRUTH VALUE* returned as *TV* when the Syntax Rules of Subclause 9.4, “Satisfaction of a <label expression> by a defined label set”, are applied with *LE* as *LABEL EXPRESSION* and the defined label set of *GE* as *DEFINED LABEL SET*.

c) If *BNT* is a <quantified path primary>, then:

i) Let *PP* be the <path primary> immediately contained in *BNT*. As a result of the transformations in the Syntax Rules, *PP* is a <parenthesized path pattern expression>. Let *R* be *RL*(*PP*) and let *S* be *SLM*(*PP*).

ii) Let *GQ* be the <general quantifier> immediately contained in *BNT*.

iii) Let *LB* be the value of the <lower bound> contained in *GQ*.

iv) Case:

1) If *GQ* contains an <upper bound>, then let *UB* be the value of the <upper bound>.

A) *RL*(*BNT*) is $R^{LB} \cup R^{LB+1} \cup \ldots \cup R^{UB-1} \cup R^{UB}$.

B) Let *TOOMUCH* be $S^{LB} \cup S^{LB+1} \cup \ldots \cup S^{UB-1} \cup S^{UB}$.

C) *SLM*(*BNT*) is the set of those path bindings in *TOOMUCH* that are consistent.

2) Otherwise:

A) *RL*(*BNT*) is $R^{LB} \cdot R^{*}$.

B) Let *WAYTOOMUCH* be $S^{LB} \cdot S^{*}$.

C) *SLM*(*BNT*) is the set of those path bindings in *WAYTOOMUCH* that are consistent.

d) If *BNT* is a <questioned path primary>, then:

i) Let *PP* be the <path primary> immediately contained in *BNT*. As a result of the transformations in the Syntax Rules, *PP* is a <parenthesized path pattern expression>. Let *R* be *RL*(*PP*) and let *S* be *SLM*(*PP*).

ii) *RL*(*BNT*) is $R^{0} \cup R$.

iii) *SLM*(*BNT*) is $S^{0} \cup S$.

e) If *BNT* is a <path primary>, then let *BNT2* be the <element pattern> or <parenthesized path pattern expression> that is immediately contained in *BNT*.

i) *RL*(*BNT*) is *RL*(*BNT2*).

ii) *SLM*(*BNT*) is *SLM*(*BNT2*).

f) If *BNT* is a <path concatenation>, then:

i) Let *PST* be the <path term> and let *PC* be the <path factor> that are immediately contained in *BNT*.

ii) $RL(BNT)$ is $RL(PST) \cdot RL(PC)$.

iii) Case:

1) If *PC* is an <edge pattern>, then $SLM(BNT)$ is $SLM(PST) \cdot SLM(PC)$

NOTE 82 — If *PC* is an <edge pattern>, then there is a <vertex pattern> to its right, which will be concatenated in a subsequent iteration of this recursion; consistency, including the directionality constraint implied by the <edge pattern>, will be checked at that point.

2) If *PC* is a <vertex pattern>, and the last <element pattern> *EP* of *PST* is an <edge pattern>, then:

NOTE 83 — By transformations in the Syntax Rules of Subclause 10.6, "<path pattern expression>", an edge binding is always immediately preceded and followed by a vertex binding. The current rule handles the situation in which the edge has been bound as the last elementary binding of *PST*, and therefore *PC* is the vertex binding that immediately follows the edge binding. Also note that the Syntax Rules have transformed every <abbreviated edge pattern> to a <full edge pattern>.

A) Let *SLMCONCAT* be $SLM(PST) \cdot SLM(PC)$.

B) For each path binding *PB* contained in *SLMCONCAT*,

I) Let GE_{right} be the vertex that is bound in the last elementary binding of *PB*.

II) Let *GE* be the edge that is bound in the penultimate elementary binding of *PB*.

III) Let GE_{left} be the vertex that is bound in the antepenultimate elementary binding of *PB*.

C) Let the propositions *L*, *U*, and *R* be defined as follows:

I) Proposition *L* is true if *GE* is a directed edge, GE_{left} is the destination vertex of *GE* and GE_{right} is the source vertex of *GE*.

II) Proposition *U* is true if *GE* is an undirected edge, and GE_{left} and GE_{right} are the vertices connected by *GE*.

III) Proposition *R* is true if *GE* is a directed edge, GE_{left} is the source vertex of *GE* and GE_{right} is the destination vertex of *GE*.

D) Let the *directionality constraint* of *EP* be

Case:

I) If *EP* is a <full edge pointing left>, then proposition *L* is true.

II) If *EP* is a <full edge undirected>, then proposition *U* is true.

III) If *EP* is a <full edge pointing right>, then proposition *R* is true.

IV) If *EP* is a <full edge left or undirected>, then proposition *L*, or proposition *U*, is true.

V) If *EP* is a <full edge undirected or right>, then proposition *U*, or proposition *R*, is true.

VI) If *EP* is a <full edge left or right>, then proposition *L*, or proposition *R*, is true.

VII) If *EP* is a <full edge any direction>, then at least one of proposition *L*, proposition *U*, or proposition *R*, is true.

E) *SLM*(*BNT*) is the set of those path bindings of *SLMCONCAT* that are consistent and satisfy the directionality constraint of *EP*.

3) Otherwise, *SLM*(*BNT*) is the set of those path bindings in *SLM*(*PST*) · *SLM*(*PC*) that are consistent.

g) If *BNT* is a <path factor>, then let *BNT2* be the <path primary>, <quantified path primary>, or <questioned path primary> immediately contained in *BNT*.

i) *RL*(*BNT*) is *RL*(*BNT2*).

ii) *SLM*(*BNT*) is *SLM*(*BNT2*).

h) If *BNT* is a <path pattern union>, then let *NMA* be the number of <path term>s immediately contained in *BNT*. Let PMO_1, ..., PMO_{NMA} be these <path term>s.

i) *RL*(*BNT*) is $RL(PMO_1) \cup RL(PMO_2) \cup ... \cup RL(PMO_{NMA})$.

ii) *SLM*(*BNT*) is $SLM(PMO_1) \cup SLM(PMO_2) \cup ... \cup SLM(PMO_{NMA})$.

i) If *BNT* is <path term>, then let *BNT2* be the <path factor> or <path concatenation> immediately contained in *BNT*.

i) *RL*(*BNT*) is *RL*(*BNT2*).

ii) *SLM*(*BNT*) is *SLM*(*BNT2*).

j) If *BNT* is a <path pattern expression>, then let *BNT2* be the <path term> or <path pattern union> immediately contained in *BNT*.

NOTE 84 — <path multiset alternation> is transformed into <path pattern union> in the Syntax Rules and therefore it is not considered separately here.

i) *RL*(*BNT*) is *RL*(*BNT2*).

ii) *SLM*(*BNT*) is *SLM*(*BNT2*).

4) Let *SM* be *SLM*(*SBI*).

5) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *SM* as *SET OF MATCHES*.

Conformance Rules

None.

9.8 Evaluation of a selective <path pattern>

Function

Evaluate a <path pattern> with a selective <path search prefix>.

Subclause Signature

```
“Evaluation of a selective <path pattern>” [General Rules] (
  Parameter: “PURE PROPERTY GRAPH”,
  Parameter: “PATH PATTERN LIST”,
  Parameter: “MACHINERY”,
  Parameter: “SELECTIVE PATH PATTERN”,
  Parameter: “INPUT SET OF LOCAL MATCHES”
) Returns: “OUTPUT SET OF LOCAL MATCHES”
```

PURE PROPERTY GRAPH — a pure property graph.

PATH PATTERN LIST — a <path pattern list>.

MACHINERY — the machinery for graph pattern matching.

SELECTIVE PATH PATTERN — a selective <path pattern>.

INPUT SET OF LOCAL MATCHES — a set, possibly infinite, of matches to SELECTIVE PATH PATTERN.

OUTPUT SET OF LOCAL MATCHES — finite subset of the INPUT SET OF LOCAL MATCHES, selected according to the criterion indicated by the selective <path search prefix> of SELECTIVE PATH PATTERN.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *PG* be the *PURE PROPERTY GRAPH*, let *PPL* be the *PATH PATTERN LIST*, let *MACH* be the *MACHINERY*, let *SEL* be the *SELECTIVE PATH PATTERN*, and let *INSLM* be the *INPUT SET OF LOCAL MATCHES* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *OUTPUT SET OF LOCAL MATCHES*.

2) It is implementation-defined (IA036) whether the General Rules of this Subclause are terminated if an exception condition is raised. If an SQL-implementation defines that it terminates execution because of an exception condition, it is implementation-dependent (UA042) which of the members of *CANDIDATES* (defined subsequently) are actually probed to establish whether they might raise an exception.

   NOTE 85 — *CANDIDATES* is potentially an infinite set, but there are algorithms to enumerate this set so as to satisfy the selection criterion of the selective <path pattern> without testing all candidate solutions. Even if the SQL-implementation defines that it terminates when an exception condition is encountered on a particular candidate solution, the order of enumerating the candidates is implementation-dependent, and it is possible that a candidate solution that would raise an exception is never tested.

3) The following components of *MACH* are identified:

   a) *ABC*, the alphabet, formed as the disjoint union of the following:

      i) *SVV*, the set of names of vertex variables.

      ii) *SEV*, the set of names of edge variables.

      iii) *SPS*, the set of subpath symbols.

      iv) *SAS*, the set of anonymous symbols.

      v) *SBS*, the set of bracket symbols.

   b) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.

4) Let *NP* be the number of <path pattern>s in *PPL*.

5) *SEL* is a selective <path pattern>. Let *j* be the bracket index of the <parenthesized path pattern expression> simply contained in *SEL*.

   NOTE 86 — "Bracket index" is defined in Subclause 9.6, "Machinery for graph pattern matching".

   NOTE 87 — By a syntactic transformation in Subclause 10.4, "<graph pattern>", this <parenthesized path pattern expression> is the entire content of *SEL* except possibly the declaration of a path variable.

6) Let *PSP* be the <path search prefix> simply contained in *SEL*.

7) Let *N* be the value of the <number of paths> or the <number of groups> specified in *PSP*. If *N* is not a positive integer, then an exception condition is raised: *data exception — invalid number of paths or groups (22G0F)* and no further General Rules of this Subclause are applied.

8) Let *p* be such that *SEL* is the *p*-th <path pattern> of *PPL*.

9) Let *CANDIDATES* be the set of every path binding *PBX* in *INSLM* such that all of the following are true:

   a) For every unconditional singleton <element variable> *EV* exposed by *SEL*, *EV* is bound to a unique graph element by the elementary bindings of *EV* contained in *PBX*.

      NOTE 88 — Anonymous symbols are not <element variable>s; there is no requirement that two anonymous symbols bind to the same graph element.

   b) For every <parenthesized path pattern expression> *PPPE* equal to or contained in *SEL*, let *i* be the bracket index of *PPPE*, and let "$[_i$" and "$]_i$" be the bracket symbols associated with *PPPE*. A *binding* of *PPPE* is a substring of *PBX* that begins with the bracket binding ("$[_i$", "$[_i$") and ends with the next bracket binding ("$]_i$", "$]_i$").

      NOTE 89 — "Bracket index" is defined in Subclause 9.6, "Machinery for graph pattern matching".

      For every binding *BPPPE* of *PPPE* contained in *PBX*, all of the following are true:

      i) For every <element variable> *EV* that is that is exposed as an unconditional singleton by *PPPE*, *EV* is bound to a unique graph element by the elementary bindings of *EV* contained in *BPPPE*.

         NOTE 90 — Anonymous symbols are not <element variable>s; there is no requirement that two anonymous symbols bind to the same graph element.

      ii) If *PPPE* contains a <parenthesized path pattern where clause> *PPPWC*, then <u>*True*</u> is the *VALUE* returned as *V* when the General Rules of Subclause 9.9, "Applying bindings to evaluate an expression", are applied with *PPL* as *GRAPH PATTERN*, the <search condition> simply contained in *PPPWC* as *EXPRESSION*, *MACH* as *MACHINERY*, *PBX* as *MULTI-PATH BINDING*, and a reference to *BPPPE* as a subset of *PBX* as *REFERENCE TO LOCAL CONTEXT*.

NOTE 91 — This is the juncture at which an exception condition might be raised. It is implementation-dependent whether to terminate if an exception condition is raised. The order of enumerating the members of *CANDIDATES* is implementation-dependent, and there is no requirement that an SQL-implementation test all candidate solutions, which can be an infinite set in any case.

10) Each path binding *PBX* of *CANDIDATES* is replaced by *REDUCE*(*PBX*).

11) Redundant duplicate path bindings are removed from *CANDIDATES*.

12) *CANDIDATES* is partitioned as follows. For every path binding *PBX* in *CANDIDATES*, the partition of *PBX* is the set of every path binding *PBY* in *CANDIDATES* such that the first and last vertex bindings of *PBX* bind the same vertices as the first and last vertex bindings, respectively, of *PBY*.

13) Each partition *PART* of *CANDIDATES* is modified as follows.

    Case:

    a) If *PSP* is an <any path search>, then

       Case:

       i) If the number of path bindings in *PART* is *N* or less, then the entire partition *PART* is retained.

       ii) Otherwise, it is implementation-dependent (UA013) which *N* path bindings of *PART* are retained.

    b) If *PSP* is a <shortest path search>, then

       Case:

       i) If *PSP* is a <counted shortest path search>, then

          Case:

          1) If the number of path bindings in *PART* is *N* or less, then the entire partition *PART* is retained.

          2) Otherwise, the path bindings of *PART* are sorted in increasing order of number of edges; the order of path bindings that have the same number of edges is implementation-dependent (US018). The first *N* path bindings in *PART* are retained.

       ii) If *PSP* is <counted shortest group search>, then the path bindings in *PART* are grouped, with each group comprising those path bindings having the same number of edges. The groups are ordered in increasing order by the number of edges.

          Case:

          1) If the number of groups in *PART* is *N* or less, then the entire partition *PART* is retained.

          2) Otherwise, the path bindings comprising the first *N* groups of *PART* are retained.

14) Let *OUTSLM* be the set of path bindings retained in *CANDIDATES* after the preceding modifications to its partitions.

15) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *OUTSLM* as *OUTPUT SET OF LOCAL MATCHES*.

Conformance Rules

None.

9.9 Applying bindings to evaluate an expression

Function

Evaluate a <value expression> or <search condition> using the bindings of graph pattern variables determined by a local context within a multi-path binding.

Subclause Signature

```
“Applying bindings to evaluate an expression” [General Rules] (
  Parameter: “GRAPH PATTERN”,
  Parameter: “EXPRESSION”,
  Parameter: “MACHINERY”,
  Parameter: “MULTI-PATH BINDING”,
  Parameter: “REFERENCE TO LOCAL CONTEXT”
) Returns: “VALUE”
```

GRAPH PATTERN — a <graph pattern>.

EXPRESSION — a <value expression> or <search condition>.

MACHINERY — the machinery for graph pattern matching.

MULTI-PATH BINDING — multi-path binding to the <graph pattern> in which to evaluate the expression.

REFERENCE TO LOCAL CONTEXT — an indication of a subset of the multi-path binding, the local context. Group bindings are confined to the local context; singleton bindings may look outside the local context.

VALUE — the evaluated value of EXPRESSION.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *GP* be the *GRAPH PATTERN*, let *EXP* be the *EXPRESSION*, let *MACH* be the *MACHINERY*, let *MPB* be the *MULTI-PATH BINDING*, and let *RTLC* be the *REFERENCE TO LOCAL CONTEXT* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *VALUE*.

2) The following components of *MACH* are identified:

   a) *ABC*, the alphabet, formed as the disjoint union of the following:

      i) *SVV*, the set of names of vertex variables.

      ii) *SEV*, the set of names of edge variables.

iii) *SPS*, the set of subpath symbols.

iv) *SAS*, the set of anonymous symbols.

v) *SBS*, the set of bracket symbols.

b) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.

3) Let *LC* be the subset of *MPB* that is indicated by *RTLC*.

NOTE 92 — The local context is passed "by reference" in order to correctly evaluate non-local singletons. For example, given the pattern:

```
( (A) -> ( (B) -> (C) WHERE A.X = B.X+C.X ) -> (D)){2}
```

then A.X makes a non-local reference to element variable A. A word of this pattern will repeat the outer <parenthesized path pattern expression> twice, requiring two evaluations of the WHERE clause. Each evaluation of the WHERE clause must locate the appropriate non-local reference to A. The bindings to the inner <parenthesized path pattern expression>, if passed "by value", might not be enough information to determine the appropriate binding to the outer <parenthesized path pattern expression>.

4) For each <element reference> *ER* that is contained in *EXP* at the same depth of graph pattern matching and that does not reference an iterator variable:

NOTE 93 — The binding of iterator variables is performed prior to invoking this Subclause in Subclause 9.11, "Applying bindings to generate a row".

a) Let *DEG* be the degree of reference of *ER*.

b) Let *EV* be the <element variable> of *ER*.

c) Case:

i) If *LC* is equal to *MPB*, then let *SPACE* be *MPB*.

NOTE 94 — That is, the search space is the entire multi-path binding. This case arises in three circumstances: 1) the evaluation of a <parenthesized path pattern where clause> in the outermost <parenthesized path pattern expression> of a selective <path pattern>; 2) the evaluation of a <graph pattern where clause>; 3) the evaluation of a <graph table column definition>.

ii) Otherwise, let *LCBI* be the bracket index of the first bracket symbol in *LC*. Let *LCPPPE* be the <parenthesized path pattern expression> contained in *GP* whose bracket index is *LCBI*. Let *PP* be the <path pattern> containing *LCPPPE*.

NOTE 95 — "Bracket index" is defined in Subclause 9.6, "Machinery for graph pattern matching".

Case:

1) If *EV* is not declared by *PP* then let *SPACE* be *MPB*.

NOTE 96 — In this case, *EV* is declared in some other <path pattern> than the one that contains *ER*. *ER* is a non-local reference, therefore *DEG* is singleton, and *EV* must be exposed as a singleton by the <path pattern>(s) that declare it.

2) If *EV* is declared by *LCPPPE*, then let *SPACE* be *LC*.

NOTE 97 — In this case, *EV* is declared locally to *LCPPPE* and the binding(s) to *ER* can be found by searching the local context *LC*.

3) Otherwise, let *DEFPPPE* be the innermost <parenthesized path pattern expression> that declares *EV* and that contains *LCPPPE*. Let *BI* be the bracket index of *DEFPPPE*. Let "$[_{BI}$" and "$]_{BI}$" be the bracket symbols whose bracket index is *BI*. Let *SPACE* be the smallest substring of *MPB* containing *LC* and beginning with "$[_{BI}$" and ending with "$]_{BI}$".

NOTE 98 — In this case, *EV* is declared in some outer scope containing *LCPPPE*, and the binding of *ER*, if any, is found by searching the innermost scope that declares *EV*. *ER* is a non-local reference, therefore *DEG* is singleton, and *EV* must be exposed as a singleton by *DEFPPPE*.

NOTE 99 — "Bracket index" is defined in Subclause 9.6, "Machinery for graph pattern matching".

d) Case:

i) If *DEG* is singleton, then

Case:

1) If there is an elementary binding *EB* of *EV* in *SPACE*, then let *LOE* be a list with a single graph element, the graph element that is bound to *EV* by *EB*.

NOTE 100 — Even if *EV* is bound multiple times in *SPACE* (expressing an equijoin on *EV*), the list has only one graph element.

2) Otherwise, let *LOE* be the empty list.

NOTE 101 — This case can only arise if *DEG* is conditional singleton.

ii) If *DEG* is group, then

Case:

1) If *SPACE* does not contain an elementary binding of *EV*, then let *LOE* be an empty list.

2) Otherwise, let *LOE* be the list of the graph elements that are bound to *EV* by elementary bindings in the order that they occur in *SPACE*, scanning *SPACE* from left to right, and retaining duplicates.

e) *LOE* is the *list of graph elements bound to ER*.

NOTE 102 — This list will be used by the General Rules of those Subclauses that evaluate *ER*, for example, Subclause 6.5, "<property reference>".

5) Let *V* be the value of *EXP*.

6) For each <element reference> *ER* contained in *EXP* at the same depth of graph pattern matching, the list of graph elements bound to *ER* is destroyed.

7) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *V* as *VALUE*.

Conformance Rules

None.

9.10 Applying bindings to evaluate a subexpression of an aggregate

Function

Evaluate a subexpression of an <aggregate function> at a particular binding of the aggregated element variable.

Subclause Signature

```
“Applying bindings to evaluate a subexpression of an aggregate” [General Rules] (
  Parameter: “BINDINGS”,
  Parameter: “AGGREGATED ELEMENT VARIABLE”,
  Parameter: “ELEMENT NUMBER”,
  Parameter: “SUBEXPRESSION”
) Returns: “VALUE”
```

BINDINGS — a set of bindings of <element reference>s.

AGGREGATED ELEMENT VARIABLE — the aggregated element variable of an <aggregate function>.

ELEMENT NUMBER — the 1-relative index into the list of graph elements that are bound by BINDINGS to AGGREGATED ELEMENT VARIABLE.

SUBEXPRESSION — a subexpression of an <aggregate function>.

VALUE — the value of the SUBEXPRESSION when AGGREGATED ELEMENT VARIABLE is bound to graph element indicated by ELEMENT NUMBER.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *B* be the *BINDINGS*, let *AEV* be the *AGGREGATED ELEMENT VARIABLE*, let *N* be the *ELEMENT NUMBER*, and let *SUBEXP* be the *SUBEXPRESSION* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *VALUE*.

2) Let *AEV* be bound to the list of graph elements GE_1, ..., GE_K in *B*.

3) Define a set of bindings *B2* as follows. For every <element reference> *ER* which is bound to a list of graph elements in *B*:

   Case:

   a) If *ER* references *AEV*, then let *ER* be bound to the graph element GE_N.

   b) Otherwise, let *ER* be bound to the same list of graph elements as it is bound to in *B*.

4) Let *V* be the value of *SUBEXP* using the bindings in *B2* and treating *AEV* as having unconditional singleton as its degree of reference.

5) *B2*, including every binding in *B2*, is destroyed.

6) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *V* as *VALUE*.

Conformance Rules

None.

9.11 Applying bindings to generate a row

Function

Generate a row of output from <graph table derived table>.

Subclause Signature

```
"Applying bindings to generate a row" [General Rules] (
  Parameter: "GRAPH PATTERN",
  Parameter: "ROWS CLAUSE",
  Parameter: "COLUMNS CLAUSE",
  Parameter: "MACHINERY",
  Parameter: "MULTI-PATH BINDING",
  Parameter: "PATH NUMBER",
  Parameter: "OFFSET"
) Returns: "ROW"
```

GRAPH PATTERN — a <graph pattern>.

ROWS CLAUSE — a <graph table rows clause>.

COLUMNS CLAUSE — a <graph table columns clause>.

MACHINERY — the machinery for graph pattern matching.

MULTI-PATH BINDING — a multi-path binding.

PATH NUMBER — 1-relative index of a <path pattern> contained in the GRAPH PATTERN (null if ROWS CLAUSE is ONE ROW PER MATCH).

OFFSET — 1-relative offset of an elementary binding contained in the path binding indicated by PATH NUMBER within MULTI-PATH BINDING.

ROW — a row to be output by <graph table>.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *GP* be the *GRAPH PATTERN*, let *RC* be the *ROWS CLAUSE*, let *CC* be the *COLUMNS CLAUSE*, let *MACH* be the *MACHINERY*, let *MPB* be the *MULTI-PATH BINDING*, let *PNUM* be the *PATH NUMBER*, and let *OFF* be the *OFFSET* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *ROW*.

   NOTE 103 — *PNUM* and *OFFSET* are only used when generating rows using <one row per iteration>, in which case *PNUM* is the number of the current path pattern being iterated over, and *OFFSET* is the 1-relative index of the compressed binding for the first iterator variable in the compressed path binding indicated by *PNUM*.

2) The following components of *MACH* are identified:

   a) *ABC*, the alphabet, formed as the disjoint union of the following:

      i) *SVV*, the set of names of vertex variables.

      ii) *SEV*, the set of names of edge variables.

      iii) *SPS*, the set of subpath symbols.

      iv) *SAS*, the set of anonymous symbols.

      v) *SBS*, the set of bracket symbols.

   b) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.

3) Let PP_1, ... PP_{NPP} be the <path pattern>s simply contained in *GP*.

4) Let *MPB* = (PP_1, ... PP_{NPP}).

5) If *RC* is <one row per iteration>, then:

   a) Case:

      i) If PP_{PNUM} declares a <path variable> *PNAME*, then *PNAME* is the *current path name*.

      ii) Otherwise, there is no current path name.

   b) Let *CPB* be the compressed path binding of PB_{PNUM}.

      NOTE 104 — The compressed path binding is defined in Subclause 9.6, “Machinery for graph pattern matching”.

   c) Let CB_1, ... CB_L be the elements of the sequence *CPB*.

   d) Case:

      i) If *RC* is <one row per vertex>, then:

         1) Let *I* be the iterator variable identified by the <vertex variable> simply contained in *RC*.

         2) The *current element number* of *I* is *OFF*.

         3) Let CB_{OFF} be the compressed binding (*LOV*, *VER*).

         4) *LOV* is the *current list of primary variable names* of *I*.

         5) For every <element reference> *ER* that references *I*, *ER* is bound to the list of graph elements whose sole element is *VER*.

      ii) If RC is <one row per step>, then:

         1) Let *I1* be the iterator variable identified by the <vertex variable 1> simply contained in *RC*, let *I2* be the iterator variable identified by the <edge variable> simply contained in *RC*, and let *I3* be the iterator variable identified by the <vertex variable 2> simply contained in *RC*.

         2) Case:

            A) If *L* is 1 (one), then:

I) The *current element number* of *I1* is *OFF*, the *current element number* of *I2* is the null value, and the *current element number* of *I3* is the null value.

II) Let CB_1 be the only element of *CPB*. CB_1 is a compressed binding (*LOV*, *VER*).

III) The *current list of primary variable names* of *I1* is *LOV*. The *current list of primary variable names* of *I2* and *I3* is empty.

IV) For every <element reference> *ER* that references *I1*, *ER* is bound to the list of graph elements whose sole member is *VER*. For every <element reference> *ERX* that references *I2* or *I3*, *ERX* is bound to an empty list of graph elements.

B) Otherwise:

I) The *current element number* of *I1* is *OFF*, the *current element number* of *I2* is the *OFF*+1, and the *current element number* of *I3* is *OFF*+2.

II) Let CB_{OFF} be the compressed binding (*LOV1*, *GE1*). Let CB_{OFF+1} be the compressed binding (*LOV2*, *GE2*). Let CB_{OFF+2} be the compressed binding (*LOV3*, *GE3*).

III) The *current list of primary variable names* of *I1* is *LOV1*. The *current list of primary variable names* of *I2* is *LOV2*. The *current list of primary variable names* of *I3* is *LOV3*.

IV) For every <element reference> *ER1* that references *I1*, *ER1* is bound to the list of graph elements whose sole member is *GE1*. For every <element reference> *ER2* that references *I2*, *ER2* is bound to the list of graph elements whose sole member is *GE2*. For every <element reference> *ER3* that references *I3*, *ER3* is bound to the list of graph elements whose sole member is *GE3*.

6) *R* is a row, with a column for each <graph table column definition> *GTCD* contained in *CC*. For each column:

a) The General Rules of Subclause 9.9, "Applying bindings to evaluate an expression", are applied with *GP* as *GRAPH PATTERN*, the <value expression> simply contained in *GTCD* as *EXPRESSION*, *MACH* as *MACHINERY*, *M* as *MULTI-PATH BINDING*, and reference to *M* as a subset of itself as *REFERENCE TO LOCAL CONTEXT*; let *VE* be the *VALUE* returned from the application of those General Rules.

b) The value of the column is *VE*.

7) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *R* as *ROW*.

Conformance Rules

None.

9.12 Creation of an element table descriptor

Function

Create an element table descriptor.

Subclause Signature

```
"Creation of an element table descriptor" [Syntax Rules] (
  Parameter: "BNFTERM"
) Returns: "DESCRIPTOR"
```

BNFTERM — a <vertex table definition> or an <edge table definition>.

DESCRIPTOR — the created element table descriptor.

Syntax Rules

1) Let *B* be the *BNFTERM* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is returned as *DESCRIPTOR*.

2) Case:

   a) If *B* simply contains <element table name> *ELTN*, then:

      i) Case:

         1) If *B* simply contains <element table alias>, then let *ETN* be that <element table alias>.

         2) Otherwise, let *ETN* be the <qualified identifier> simply contained in *ELTN*.

      ii) Let *ET* be the table identified by *ELTN*.

      iii) *ET* shall be either a persistent base table or a viewed table.

   b) Otherwise:

      i) Let *ETN* be the <element table alias>.

      ii) Let *ETQE* be the <query expression> simply contained in the <table subquery> immediately contained in *B*.

      iii) No <table reference> generally contained in *ETQE* shall identify any declared local temporary table.

      iv) No <table reference> generally contained in *ETQE* shall contain a <data change delta table>.

      v) Let *ET* be the table specified by *ETQE*.

3) *ETN* is a range variable for *ET* whose scope is *B*.

4) Case:

   a) If <element table key clause> is not specified, then

      Case:

i) If the table descriptor of *ET* includes a unique constraint descriptor *UCD* that specifies PRIMARY KEY, then let *ETK* be the list of the names of the unique columns included in *UCD* in order of their ordinal position in *ET*.

ii) Otherwise, the table descriptor of *ET* shall include a preferred candidate key. Let *ETK* be the list of columns in that preferred candidate key in order of their ordinal position in *ET*.

b) Otherwise, let *ETK* be the <column name list> simply contained in <element table key clause>.

i) No <column name> simply contained in *ETK* shall be equivalent to another <column name> simply contained in *ETK*.

ii) Each <column name> simply contained in *ETK* shall identify a column of *ET*.

5) If *B* does not simply contain an <element table label and properties clause>, then the <element table label and properties clause> DEFAULT LABEL PROPERTIES ARE ALL COLUMNS is implicit.

6) If *B* simply contains an <element table label and properties clause> that immediately contains an <element table properties clause> *ETPC*, then *ETPC* is effectively replaced with a <label and properties> DEFAULT LABEL *ETPC*.

7) An implicit or explicit <element table label clause> that specifies DEFAULT LABEL is effectively replaced with an <element table label clause> LABEL *ETN*.

8) Let *NLP* be the number of explicit or implicit <label and properties>s simply contained in <element table label and properties clause> *ETLPC* simply contained in *B*.

9) For each k, 1 (one) $\leq k \leq NLP$, let LP_k be the k-th <label and properties>:

a) Let ETL_k be the <label name> simply contained in LP_k. ETL_k shall not be equivalent to any other <label name> simply contained in *ETLPC*.

NOTE 105 — The preceding Syntax Rule ensures that no two labels within the same element table have the same name.

b) Case:

i) If NO PROPERTIES is specified, then let NP_k be 0 (zero).

ii) If LP_k simply contains an <element table parenthesized derived property list>, then let NP_k be the number of simply contained <derived property>s. For i, 1 (one) $\leq i \leq NP_k$:

1) Every <column reference> simply contained in the i-th <derived property> shall reference a column of *ET*.

2) Case:

A) If the i-th <derived property> simply contains a <property name>, then let $PN_{k,i}$ be that <property name>.

B) Otherwise, the <value expression> simply contained in the i-th <derived property> shall be a <column reference>. Let $PN_{k,i}$ be the name of the column referenced by that <column reference>.

3) Let $PT_{k,i}$ be the declared type of the <value expression> simply contained in the i-th <derived property>.

4) Let $PV_{k,i}$ be the <value expression> simply contained in the i-th <derived property>.

5) $PV_{k,i}$ shall not contain a potential source of non-determinism.

6) $PV_{k,i}$ shall not generally contain a <routine invocation> whose subject routine possibly reads SQL-data.

7) $PV_{k,i}$ shall not contain a <query expression>.

iii) If EXCEPT is specified, then let $NETC_k$ be the number of columns of *ET*, let EC_k be the number of <column name>s simply contained in the <except column name list> $ECNL_k$, and let NP_k be $NETC_k - EC_k$.

1) No <column name> simply contained in $ECNL_k$ shall be equivalent to any other <column name> simply contained in $ECNL_k$.

2) Each <column name> simply contained in $ECNL_k$ shall identify a column of *ET*.

3) Let *RC* be the set of columns of *ET* that are not identified by a <column name> simply contained in $ECNL_k$ (there are NP_k such columns).

4) For *i*, 1 (one) $\leq i \leq NP_k$:

A) Let $PN_{k,i}$ be the name of the *i*-th column in *RC*.

B) Let $PT_{k,i}$ be the declared type of the *i*-th column in *RC*.

C) Let $PV_{k,i}$ be the name of the *i*-th column in *RC*.

iv) Otherwise, let NP_k be the number of columns of *ET*. For *i*, 1 (one) $\leq i \leq NP_k$:

1) Let $PN_{k,i}$ be the name of the *i*-th column of *ET*.

2) Let $PT_{k,i}$ be the declared type of the *i*-th column of *ET*.

3) Let $PV_{k,i}$ be the name of the *i*-th column of *ET*.

c) For 1 (one) $\leq s \leq NP_k$, 1 (one) $\leq t \leq NP_k$, $s \neq t$, $PN_{k,s}$ shall not be equivalent to $PN_{k,t}$.

NOTE 106 — The preceding Syntax Rule ensures that no two properties within the same label have the same name.

10) For each pair of properties ($P_{k,s}$, $P_{l,t}$), 1 (one) $\leq k \leq NLP$, 1 (one) $\leq l \leq NLP$, $k \neq l$, 1 (one) $\leq s \leq NP_k$, 1 (one) $\leq t \leq NP_l$, if $PN_{k,s}$ is equivalent to $PN_{l,t}$, then $PV_{k,s}$ shall have the same left normal form derivation as $PV_{l,t}$.

NOTE 107 — "Left normal form derivation" is defined in Subclause 6.2, "Notation provided in the ISO/IEC 9075 series", in ISO/IEC 9075-1.

NOTE 108 — The preceding Syntax Rule ensures that two properties with the same name within the same element table but in different labels are the same; i.e., have the same definition and, by implication, the same declared type.

11) Let *D* be an element table descriptor that includes:

a) Case:

i) If *ET* is a persistent base table or a viewed table, then the schema-qualified name of *ET*: *ELTN*.

ii) Otherwise, the <query expression>: *ETQE*.

b) The element table alias: *ETN*.

c) The list of columns uniquely identifying a row in *ET*: *ETK*.

d) A set of labels. For each label, 1 (one) $\leq k \leq NLP$:

   i) The name of the label: ETL_k.

   ii) The set of properties associated with the label. For each property, 1 (one) $\leq i \leq NP_k$:

      1) The name of the property: $PN_{k,i}$.

      2) The declared type of the property: $PT_{k,i}$.

      3) The value expression associated with the property: $PV_{k,i}$.

12) Evaluation of the Syntax Rules is terminated and control is returned to the invoking Subclause, which receives *D* as *DESCRIPTOR*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.13 Creation of a vertex table descriptor

Function

Create a vertex table descriptor.

Subclause Signature

```
“Creation of a vertex table descriptor” [Syntax Rules] (
  Parameter: “BNFTERM”
) Returns: “DESCRIPTOR”
```

BNFTERM — a <vertex table definition>.

DESCRIPTOR — the created vertex table descriptor.

Syntax Rules

1) Let *B* be the *BNFTERM* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is returned as *DESCRIPTOR*.
2) *B* shall not simply contain <source vertex table> or <destination vertex table>.
3) The Syntax Rules of Subclause 9.12, “Creation of an element table descriptor”, are applied with *B* as *BNFTERM*; let *V* be the *DESCRIPTOR* returned from the application of those Syntax Rules.
4) *V* is a vertex table descriptor.
5) Evaluation of the Syntax Rules is terminated and control is returned to the invoking Subclause, which receives *V* as *DESCRIPTOR*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.14 Creation of an edge table descriptor

Function

Create an edge table descriptor.

Subclause Signature

```
"Creation of an edge table descriptor" [Syntax Rules] (
  Parameter: "BNFTERM",
  Parameter: "SOURCE",
  Parameter: "DESTINATION"
) Returns: "DESCRIPTOR"
```

BNFTERM — an <edge table definition>.

SOURCE — the vertex table descriptor of the source of the edge.

DESTINATION — the vertex table descriptor of the desctination of the edge.

DESCRIPTOR — the created edge table descriptor.

Syntax Rules

1) Let *B* be the *BNFTERM*, let *SVTD* be the *SOURCE*, and let *DVTD* be the *DESTINATION* in an application of the Syntax Rules of this Subclause. The result of the application of this Subclause is returned as *DESCRIPTOR*.

2) *B* shall simply contain a <source vertex table> *SVT* and a <destination vertex table> *DVT*.

3) If *SVT* does not simply contain a <source vertex table key clause>, then *SVT* shall not simply contain a <referenced source column list>. If *SVT* simply contains a <source vertex table key clause>, then *SVT* shall also simply contain a <referenced source column list>.

4) If *DVT* does not simply contain a <destination vertex table key clause>, then *DVT* shall not simply contain a <referenced destination column list>. If *DVT* simply contains a <destination vertex table key clause>, then *DVT* shall also simply contain a <referenced destination column list>.

5) The <source vertex table alias> *SA* simply contained in *SVT* shall be equivalent to the element table alias included in *SVTD*. The <destination vertex table alias> *DA* simply contained in *DVT* shall be equivalent to the element table alias included in *DVTD*.

6) Let *SV* be the table identified by *SA*.

7) Let *DV* be the table identified by *DA*.

8) Case:

   a) If *B* simply contains an <element table name> *ETN*, then let *ET* be the table identified by *ETN*.

   b) Otherwise, let *ET* be the table specified by the <query expression> simply contained in the <table subquery> simply contained in *B*.

9) Case:

   a) If *SVT* does not simply contain a <source vertex table key clause>, then the table descriptor of *ET* shall include exactly one referential constraint descriptor *RCD* whose referenced table is *SV*.

i) Let *ESK* be the list of column names of the referencing columns included in *RCD*.

ii) Let *SVK* be the list of column names of the referenced columns included in *RCD*.

b) Otherwise:

i) Let *ESK* be the <column name list> simply contained in the <source vertex table key clause>.

1) No <column name> simply contained in *ESK* shall be equivalent to any other <column name> simply contained in *ESK*.

2) Each <column name> simply contained in *ESK* shall identify a column of *ET*.

ii) Let *SVK* be the <column name list> simply contained in the <referenced source column list>.

1) No <column name> simply contained in *SVK* shall be equivalent to any other <column name> simply contained in *SVK*.

2) Each <column name> simply contained in *SVK* shall identify a column of *SV*.

10) Case:

a) If *DVT* does not simply contain a <destination vertex table key clause>, then the table descriptor of *ET* shall include exactly one referential constraint descriptor *RCD* whose referenced table is *DV*.

i) Let *EDK* be the list of column names of the referencing columns included in *RCD*.

ii) Let *DVK* be the list of column names of the referenced columns included in *RCD*.

b) Otherwise:

i) Let *EDK* be the <column name list> simply contained in the <destination vertex table key clause>.

1) No <column name> simply contained in *EDK* shall be equivalent to any other <column name> simply contained in *EDK*.

2) Each <column name> simply contained in *EDK* shall identify a column of *ET*.

ii) Let *DVK* be the <column name list> simply contained in the <referenced destination column list>.

1) No <column name> simply contained in *DVK* shall be equivalent to any other <column name> simply contained in *DVK*.

2) Each <column name> simply contained in *DVK* shall identify a column of *DV*.

11) The Syntax Rules of Subclause 9.12, "Creation of an element table descriptor", are applied with *B* as *BNFTERM*; let *D* be the *DESCRIPTOR* returned from the application of those Syntax Rules.

12) Let *E* be an edge table descriptor that is a copy of *D* and additionally includes:

a) The name the source vertex table: *SA*.

b) The edge source key: *ESK*.

c) The source vertex key: *SVK*.

d) The name of the destination vertex table: *DA*.

e) The edge destination key: *EDK*.

f) The destination vertex key: *DVK*.

13) Evaluation of the Syntax Rules is terminated and control is returned to the invoking Subclause, which receives *E* as *DESCRIPTOR*.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.15 Consistency check of a tabular property graph descriptor

Function

Check the consistency of a tabular property graph descriptor.

Subclause Signature

```
"Consistency check of a tabular property graph descriptor" [Syntax Rules] (
  Parameter: "TPGDESCRIPTOR"
)
```

TPGDESCRIPTOR — the tabular property graph descriptor to be checked.

Syntax Rules

1) Let *TPGD* be the *TPGDESCRIPTOR* in an application of the Syntax Rules of this Subclause.

2) Let *n* be the number of vertex table descriptors included in *TPGD*.

3) Let *m* be the number of edge table descriptors included in *TPGD*.

4) For *i*, 1 (one) ≤ *i* ≤ *n*+*m*:

   a) Let $DESC_i$ be the *i*-th element table descriptor included in *TPGD*. $DESC_i$ is either a vertex table descriptor or an edge table descriptor.

   b) The element table alias included in $DESC_i$ shall not be equivalent to any element table alias included in any other element table descriptor (which is either a vertex or an edge table descriptor) included in *TPGD*.

   c) For each label *L* included in $DESC_i$:

      i) Let *LN* be the name of *L*.

      ii) If a label *LL* included in any other element table descriptor (which is either a vertex or an edge table descriptor) included in *TPGD* has a name that is equivalent to *LN*, then *L* and *LL* shall have the same number of properties and the name of each property of *L* shall be equivalent to the name of the corresponding property of *LL*.

      iii) For each property *P* of *L*:

         1) Let *PN* be the name of *P*.

         2) If another property *PP* of any other label included in any other element table descriptor included in *TPGD* has a name that is equivalent to *PN*, then the declared type of *PP* shall be the same as the declared type of *P*.

            NOTE 109 — The preceding Syntax Rule ensures the consistency of the declared types of all properties with the same name across all labels of all vertex and all edge tables.

5) For each edge table descriptor ETD_i, 1 (one) ≤ *i* ≤ *m*, included in *TPGD*:

   a) The name of the source vertex table included in ETD_i shall be equivalent to the element table alias included in some vertex table descriptor included in *TPGD*.

b) The name of the destination vertex table included in *ETD_i* shall be equivalent to the element table alias included in some vertex table descriptor included in *TPGD*.

6) Evaluation of the Syntax Rules is terminated and control is returned to the invoking Subclause.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

9.16 Deriving a pure property graph descriptor from a tabular property graph descriptor

Function

Derive a pure property graph descriptor from a tabular property graph descriptor.

Subclause Signature

```
"Deriving a pure property graph descriptor from a tabular property graph descriptor"
[General Rules] (
  Parameter: "TABULAR PROPERTY GRAPH DESCRIPTOR"
) Returns: "PURE PROPERTY GRAPH DESCRIPTOR"
```

TABULAR PROPERTY GRAPH DESCRIPTOR — a tabular property graph descriptor.

PURE PROPERTY GRAPH DESCRIPTOR — the derived pure property graph descriptor.

Syntax Rules

None.

Access Rules

None.

General Rules

1) Let *TPGD* be the *TABULAR PROPERTY GRAPH DESCRIPTOR* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *PURE PROPERTY GRAPH DESCRIPTOR*.

2) Let *PPGD* be a pure property graph descriptor derived from *TPGD* as follows:

   NOTE 110 — If *TPGD* is empty, then *PPGD* is empty as well.

   a) The labels of *PPGD* are the unique labels of every vertex and edge table included in *TPGD*. For each label, this includes the label name and the properties (including name and declared type) of the corresponding label in *TPGD*.

   b) The set of defined vertex label sets of *PPGD* comprises, for each vertex table *V* of *TPGD*, the set of labels of *V*.

   c) The set of edge defined label sets of *PPGD* comprises, for each edge table *E* of *TPGD*, the set of labels of *E*. Each edge defined label set has an indication that the edges having that label set are always directed.

   d) *PPGD* includes a set of edge triplets. Each edge triplet has an indication that it describes directed edges, and consists of an edge triplet defined label set *EL*, a source vertex defined label set *SL*, and a destination defined vertex label set *DL*. For each edge table *ET*, *EL* consists of every label associated with *ET*, *SL* consists of every label associated with *ET*'s source vertex table, and *DL* consists of every label associated with *ET*'s destination vertex table.

3) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *PPGD* as *PURE PROPERTY GRAPH DESCRIPTOR*.

Conformance Rules

None.

10 Additional common elements

This Clause modifies Clause 10, “Additional common elements”, in ISO/IEC 9075-2.

10.1 <aggregate function>

This Subclause modifies Subclause 10.9, “<aggregate function>”, in ISO/IEC 9075-2.

Function

Augment the 1st paragraph by adding “or a list of graph elements” to the end of the sentence.

Format

```
<aggregate function> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <element variable count function> [ <filter clause> ]

<element variable count function> ::=
  BINDING_COUNT <left paren> <element reference> <right paren>
```

Syntax Rules

1) Insert after SR 14): An <element reference> simply contained in an <element variable count function> is an *aggregated argument* of *SFE*.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 1): If *AF* is immediately contained in a within-match aggregate *WMA*, then

   a) Let *B* be the set of bindings of <element reference>s (that is, the set of functions that map <element reference>s to lists of graph elements) determined by the invocation of the General Rules of Subclause 9.9, “Applying bindings to evaluate an expression”, during which *AF* is being evaluated.

      NOTE 111 — The evaluation of a within-match aggregate necessarily occurs during an invocation of Subclause 9.9, “Applying bindings to evaluate an expression”.

   b) Let *SUBEXPS* be the set of <property reference>s, <graph element predicate>s and <graphical value expression primary>s contained in *AF*. Let *SN* be the cardinality of *SUBEXPS*. Let SE_1, ..., SE_{SN} be an enumeration of the elements of *SUBEXPS*.

   c) Let CN_1, ..., CD_{SN}, *ELNO* be a list of *SN*+1 column names that are mutually distinct, and distinct from any column names contained in *AF*.

d) Let *AEV* be the aggregated element variable of *WMA*. Let *EN* be the number of graph elements that are bound to *AEV* in *B*.

e) Let *TZ* be a table with *EN* rows and *SN*+1 columns, determined as follows:

   i) For all *i*, $1 \leq i \leq SN$, the name of the *i*-th column is CN_i and the declared type of the *i*-th column is the declared type of SE_i.

   ii) The name of the (*SN*+1)-th column is *ELNO* and its declared type is exact numeric with scale 0 (zero).

   iii) For all *j*, 1 (one) $\leq j \leq EN$, there is a row in *TZ* whose columns are determined as follows:

      1) For all *i*, $1 \leq i \leq SN$, the General Rules of Subclause 9.10, "Applying bindings to evaluate a subexpression of an aggregate", are applied with *B* as *BINDINGS*, *AEV* as *AGGREGATED ELEMENT VARIABLE*, *j* as *ELEMENT NUMBER*, and SE_i as *SUBEXPRESSION*; let *V* be the *VALUE* returned from the application of those General Rules The value of the *i*-th column of *TZ* is *V*.

      2) The value of the (*SN*+1)-th column of *TZ* is *j*.

   iv) In *AF*, for all *i*, $1 \leq i \leq SN$, SE_i is effectively replaced by CN_i.

   v) If *AF* contains <path order clause> *POC*, then let *DIR* be ASC or DESC that is specified or implied by *POC*. *POC* is effectively replaced by

```
ORDER BY ELNO DIR
```

2) Insert after GR 1)c): If *AF* is immediately contained in a within-match aggregate, then *TZ*.

3) Insert after GR 6): If <element variable count function> is specified, then the result is the cardinality of *T1*.

Conformance Rules

1) Insert after the last CR: Without Feature G820, "BINDING_COUNT", conforming SQL language shall not contain an <element variable count function>.

10.2 <sort specification list>

This Subclause modifies Subclause 10.10, “<sort specification list>”, in ISO/IEC 9075-2.

Function

Specify a sort order.

Format

```
<sort specification list> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <path order clause>

<path order clause> ::=
  PATH ORDER [ <ordering specification> ]
```

Syntax Rules

1) Insert after the last SR: If <path order clause> *POC* is specified, then:

   a) *POC* shall be simply contained in a within-match aggregate which is one of the following: <array aggregate function>, <listagg set function>, <JSON array aggregate constructor>, or <XML aggregate>.

   b) If <ordering specification> is not specified, then ASC is implicit.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

NOTE 112 — <path order clause> is transformed into a <sort specification> by the General Rules of Subclause 10.1, “<aggregate function>”, so no rules are required here.

Conformance Rules

1) Insert after the last CR: Without Feature G840, “Path-ordered aggregates”, conforming SQL language shall not contain <path order clause>.

10.3 <graph reference>

Function

Reference an SQL-property graph.

Format

```
<graph reference> ::=
  <property graph name>
```

Syntax Rules

1) The <schema qualified name> contained in <property graph name> shall identify an SQL-property graph.

Access Rules

None.

General Rules

None.

Conformance Rules

None.

10.4 <graph pattern>

Function

Specify a pattern to be matched in a property graph.

Subclause Signature

```
"<graph pattern>" [General Rules] (
  Parameter: "PURE PROPERTY GRAPH",
  Parameter: "GRAPH PATTERN",
  Parameter: "PATH PATTERN LIST",
  Parameter: "MACHINERY"
) Returns: "SET OF REDUCED MATCHES"
```

PURE PROPERTY GRAPH — a pure property graph

GRAPH PATTERN — a <graph pattern>.

PATH PATTERN LIST — a <path pattern list>.

MACHINERY — the machinery for graph pattern matching.

SET OF REDUCED MATCHES — the resulting set of reduced matches.

Format

```
<graph pattern> ::=
  [ <match mode> ] <path pattern list>
      [ <keep clause> ]
      [ <graph pattern where clause> ]

<match mode> ::=
    <repeatable elements match mode>
  | <different edges match mode>

<repeatable elements match mode> ::=
  REPEATABLE <element bindings or elements>

<different edges match mode> ::=
  DIFFERENT <edge bindings or edges>

<element bindings or elements> ::=
    ELEMENT [ BINDINGS ]
  | ELEMENTS

<edge bindings or edges> ::=
    <edge synonym> [ BINDINGS ]
  | <edges synonym>

<path pattern list> ::=
  <path pattern> [ { <comma> <path pattern> }... ]

<path pattern> ::=
  [ <path variable declaration> ] [ <path pattern prefix> ] <path pattern expression>

<path variable declaration> ::=
  <path variable> <equals operator>
```

```
<keep clause> ::=
  KEEP <path pattern prefix>

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

Syntax Rules

1) Let *GP* be the <graph pattern>. Let *GT* be the <graph table> that simply contains *GP*.

2) If *BNF1* and *BNF2* are instances of two BNF non-terminals, both contained in *GP* without an intervening <graph pattern>, then *BNF1* and *BNF2* are said to be *at the same depth of graph pattern matching*.

   NOTE 113 — *BNF1* can contain *BNF2* while being at the same depth of graph pattern matching.

3) In a <path pattern list>, if two <path pattern>s expose an element variable *EV*, then both shall expose *EV* as an unconditional singleton variable.

   NOTE 114 — This case expresses an implicit join on *EV*. Implicit joins between conditional singleton variables or group variables are forbidden.

4) Two <path pattern>s shall not expose the same subpath variable.

   NOTE 115 — Implicit equijoins on subpath variables are not supported.

5) The name of a vertex variable shall not be equivalent to the name of an edge variable declared at the same depth of graph pattern matching.

6) If <keep clause> *KP* is specified, then:

   a) Let *PSP* be the <path pattern prefix> simply contained in *KP*.

   b) For each <path pattern> *PP* simply contained in *GP*:

      i) *PP* shall not contain a <path search prefix>.

      ii) Case:

         1) If *PP* specifies a <path variable declaration>, then let *PVDECL* be that <path variable declaration>.

         2) Otherwise, let *PVDECL* be the zero-length string.

      iii) Case:

         1) If *PP* specifies a <path mode prefix>, then let *PMP* be that <path mode prefix>.

         2) Otherwise, let *PMP* be the zero-length string.

      iv) Let *PPE* be the <path pattern expression> simply contained in *PP*.

      v) *PP* is replaced by

         ```
         PVDECL PSP ( PMP PPE )
         ```

   c) The <keep clause> is removed from the <graph pattern>.

7) After the preceding transformations, for every <path pattern> *PP*, if *PP* contains a <path pattern prefix> *PPP* that specifies a <path mode> *PM*, then

   a) Case:

i) If *PP* specifies a <path variable declaration>, then let *PVDECL* be that <path variable declaration>.

ii) Otherwise, let *PVDECL* be the zero-length string.

b) Let *PPE* be the <path pattern expression> simply contained in *PP*.

c) *PP* is replaced by

```
PVDECL PPP ( PM PPE )
```

NOTE 116 — One effect of the preceding transforms is that every <path mode> expressed outside a <parenthesized path pattern expression> is also expressed within a <parenthesized path pattern expression>. For example,

```
ALL SHORTEST TRAIL GROUP <path pattern expression>
```

is rewritten as

```
ALL SHORTEST TRAIL GROUP ( TRAIL <path pattern expression> )
```

The TRAIL specified outside the parentheses is now redundant. The benefit is that the definition of a consistent path binding in Subclause 9.6, "Machinery for graph pattern matching", only has to consider <path mode>s declared in <parenthesized path pattern expression>s.

8) Let *GPT* be the <graph pattern> after the preceding syntactic transformations.

9) Let *PPL* be the <path pattern list> simply contained in *GPT*.

10) If *GPT* does not specify a <match mode>, then an implementation-defined (ID022) <match mode> is implicit.

11) Let *MM* be the <match mode> implicitly or explicitly specified by *GPT*.

12) If *MM* is <different edges match mode> and *PPL* simply contains a <path pattern> that is selective, then *PPL* shall not simply contain any other <path pattern>.

NOTE 117 — If *MM* is <different edges match mode> and there is a selective <path pattern> *SPP*, then *PPL* must only contain *SPP*. If there is no selective <path pattern> in *GPT*, then there are no restrictions on how many non-selective <path pattern>s are contained in *PPL*. If *MM* is <repeatable elements match mode>, then there is no restriction on how many (selective and non-selective) <path pattern>s are contained in *PPL*.

13) Let *E* be an element variable declared by *GPT*.

Case:

a) If *E* is exposed by *GPT* as an unconditional singleton, then *E* is a *global unconditional singleton* of *GPT*.

b) Otherwise, let *PPPE* be the outermost <parenthesized path pattern expression> that exposes *E* as an unconditional singleton; the *unconditional singleton scope index* of *E* in *GPT* is the bracket index of *PPPE*.

NOTE 118 — Bracket index is defined in Subclause 9.6, "Machinery for graph pattern matching". The unconditional singleton scope index is well-defined because implicit equijoins between conditional singleton variables or group variables are forbidden. Hence there cannot be two <parenthesized path pattern expression>s that expose *E* as a conditional singleton or group variable unless one is contained in the other. For example,

```
( ( ( -[E]-> ) -[E]-> )* -[F]-> )*
```

The unconditional singleton scope of E is the middle <parenthesized path pattern expression> in the nest of three.

14) After the preceding transformations, for every <quantified path primary> *QPP* contained in *GPT*, at least one of the following shall be true:

a) The <graph pattern quantifier> of *QPP* is bounded.

b) *QPP* is contained in a restrictive <parenthesized path pattern expression>.

c) *QPP* is contained in a selective <path pattern>.

d) *MM* is <different edges match mode>.

NOTE 119 — Unless an explicit <path mode> other than WALK is specified, an implicit or explicit specification of a <different edges match mode> effectively imparts the <path mode> TRAIL without the presence of the TRAIL keyword.

15) Each <path variable> *PV* contained in *GPT* is the name of a *path variable*. The *degree of exposure* of the path variable that *PV* identifies is unconditional singleton.

Access Rules

None.

General Rules

1) Let *PG* be the *PURE PROPERTY GRAPH*, let *GPT* be the *GRAPH PATTERN*, let *PPL* be the *PATH PATTERN LIST*, and let *MACH* be the *MACHINERY* in an application of the General Rules of this Subclause. The result of the application of this Subclause is returned as *SET OF REDUCED MATCHES*.

2) The following components of *MACH* are identified:

a) *ABC*, the alphabet, formed as the disjoint union of the following:

i) *SVV*, the set of names of vertex variables.

ii) *SEV*, the set of names of edge variables.

iii) *SPS*, the set of subpath symbols.

iv) *SAS*, the set of anonymous symbols.

v) *SBS*, the set of bracket symbols.

b) *REDUCE*, the function mapping path bindings to path bindings, and multi-path bindings to multi-path bindings.

3) Let *NP* be the number of <path pattern>s simply contained in *PPL*. Let PP_1, ..., PP_{NP} be the <path pattern>s simply contained in *PPL* after the transformations in the Syntax Rules.

4) A multi-path binding *MPBINDING* is *different-edges-matched* if, for every edge binding *EB1* = (*EV1*, *E*) contained in *MPBINDING*, there is no edge binding *EB2* = (*EV2*, *E*) contained in *MPBINDING* at a different position than *EB1* that binds the edge *E*.

5) For every *i*, 1 (one) ≤ *i* ≤ *NP*:

a) Let *PPE* be the <parenthesized path pattern expression> simply contained in PP_i.

b) The General Rules of Subclause 9.7, "Evaluation of a <path pattern expression>", are applied with *PG* as *PURE PROPERTY GRAPH*, *PPL* as *PATH PATTERN LIST*, *MACH* as *MACHINERY*, and *PPE* as *SPECIFIC BNF INSTANCE*; let $SMPPE_i$ be the *SET OF MATCHES* returned from the application of those General Rules.

NOTE 120 — If an elementary variable has been multiply declared within a restrictive <parenthesized path pattern expression> *PP*, then no matches are returned for *PP*. For example:

```
MATCH ACYCLIC (X) -> (X)
```

does not find any results, even if there are vertices with self-edges.

c) Case:

i) If PP_i is a selective <path pattern>, then

1) Case:

A) If *MM* is <different edges match mode>, then let $SMUP_i$ be the set of different-edges-matched multi-path bindings in $SMPPE_i$.

NOTE 121 — If an edge variable has been multiply declared within a <path pattern> *PP*, then no matches are returned for *PP*. For example, the following produces no results:

```
MATCH DIFFERENT EDGES
      ANY SHORTEST () -[E]-> () -[E]-> ()
```

2) Otherwise, let $SMUP_i$ be $SMPPE_i$.

ii) The General Rules of Subclause 9.8, "Evaluation of a selective <path pattern>", are applied with *PG* as *PURE PROPERTY GRAPH*, *PPL* as *PATH PATTERN LIST*, *MACH* as *MACHINERY*, PP_i as *SELECTIVE PATH PATTERN*, and $SMUP_i$ as *INPUT SET OF LOCAL MATCHES*; let SM_i be the *OUTPUT SET OF LOCAL MATCHES* returned from the application of those General Rules.

d) Otherwise, let SM_i be $SMPPE_i$.

6) Let *CROSS* be the cross product $SM_1 \times \dots \times SM_{NP}$.

7) Let *INNER* be the set of multi-path bindings *MPB* in *CROSS* such that, for every unconditional singleton <element variable> *USV* exposed by *PPL*, *USV* is bound to a unique graph element by the elementary bindings of *USV* contained in *MPB*.

NOTE 122 — Anonymous symbols are not <element variable>s; there is no requirement that two anonymous symbols bind to the same graph element.

8) Case:

a) If *MM* is <different edges match mode>, then let *BINDINGS* be the set of different-edges-matched multi-path bindings in *INNER*.

NOTE 123 — If an edge variable has been multiply declared within a <graph pattern> *GP*, then no matches are returned for *GP*. For example, the following produces no results:

```
MATCH DIFFERENT EDGES () -[E]-> (), () -[E]-> ()
```

and neither does the following:

```
MATCH DIFFERENT EDGES () -[E]-> () -[E]-> ()
```

b) Otherwise let *BINDINGS* be *INNER*.

9) A *match* of *GPT* is a multi-path binding $M = (PB_1, \dots, PB_{NP})$ of *NP* path bindings in *BINDINGS*, such that all of the following are true:

a) For every *j*, 1 (one) $\leq j \leq NP$, and for or every <parenthesized path pattern expression> *PPPE* contained in PP_j, let *i* be the bracket index of *PPPE*, and let $"[_i"$ and $"]_i"$ be the bracket symbols associated with *PPPE*. A *binding* of *PPPE* is a substring of PB_j that begins with the bracket binding ("$[_i$", "$[_i$") and ends with the next bracket binding ("$]_i$", "$]_i$").

NOTE 124 — "Bracket index" is defined in Subclause 9.6, "Machinery for graph pattern matching".

For every binding *BPPPE* of *PPPE* contained in PB_j, all of the following are true:

i) For every <element variable> *EV* that is exposed as an unconditional singleton by *PPPE*, *EV* is bound to a unique graph element by the elementary bindings of *EV* contained in *BPPPE*.

NOTE 125 — Anonymous symbols are not <element variable>s; there is no requirement that two anonymous symbols bind to the same graph element.

ii) If *PPPE* contains a <parenthesized path pattern where clause> *PPPWC*, then *True* is the *VALUE* returned as *V1* when the General Rules of Subclause 9.9, "Applying bindings to evaluate an expression", are applied with *GPT* as *GRAPH PATTERN*, the <search condition> simply contained in *PPPWC* as *EXPRESSION*, *MACH* as *MACHINERY*, *M* as *MULTI-PATH BINDING*, and a reference to *BPPPE* as *REFERENCE TO LOCAL CONTEXT*.

b) If *GPT* contains a <graph pattern where clause> *GPWC*, then *True* is the *VALUE* returned as *V2* when the General Rules of Subclause 9.9, "Applying bindings to evaluate an expression", are applied with *GPT* as *GRAPH PATTERN*, *GPWC* as *EXPRESSION*, *MACH* as *MACHINERY*, *M* as *MULTI-PATH BINDING*, and a reference to *M* as *REFERENCE TO LOCAL CONTEXT*.

10) A *reduced match* $RM = (RPB_1, ..., RPB_{NP})$ is obtained from a match $M = (PB_1, ..., PB_{NP})$ as $RM = REDUCE(M)$.

11) Let *SRM* be the set of reduced matches.

NOTE 126 — Set-theoretic deduplication will occur here. That is, two or more matches can reduce to the same reduced match; this scenario is regarded as contributing only a single reduced match to the result set.

12) Evaluation of the General Rules is terminated and control is returned to the invoking Subclause, which receives *SRM* as *SET OF REDUCED MATCHES*.

Conformance Rules

1) Without Feature G000, "Graph pattern", conforming SQL language shall not contain a <graph pattern>.

2) Without Feature G001, "Repeatable-elements match mode", conforming SQL language shall not contain a <repeatable elements match mode>.

3) Without Feature G002, "Different-edges match mode", conforming SQL language shall not contain a <different edges match mode>.

4) Without Feature G003, "Explicit REPEATABLE ELEMENTS keyword", conforming SQL language shall not contain a <match mode> that specifies REPEATABLE ELEMENTS or REPEATABLE ELEMENT BINDINGS.

5) Without Feature G004, "Path variables", conforming SQL language shall not contain a <path pattern> that simply contains a <path variable declaration>.

6) Without Feature G005, "Path search prefix in a path pattern", conforming SQL language shall not contain a <path pattern> that simply contains a <path pattern prefix> that is a <path search prefix>.

7) Without Feature G006, "Graph pattern KEEP clause: path mode prefix", conforming SQL language shall not contain a <keep clause> that simply contains a <path mode prefix>.

8) Without Feature G007, "Graph pattern KEEP clause: path search prefix", conforming SQL language shall not contain a <keep clause> that simply contains a <path search prefix>.

9) Without Feature G008, "Graph pattern WHERE clause", conforming SQL language shall not contain a <graph pattern where clause>.

10.5 <path pattern prefix>

Function

Specify a path-finding operation and a path mode.

Format

```
<path pattern prefix> ::=
    <path mode prefix>
  | <path search prefix>

<path mode prefix> ::=
  <path mode> [ <path or paths> ]

<path mode> ::=
    WALK
  | TRAIL
  | SIMPLE
  | ACYCLIC

<path search prefix> ::=
    <all path search>
  | <any path search>
  | <shortest path search>

<all path search> ::=
  ALL [ <path mode> ] [ <path or paths> ]

<path or paths> ::=
    PATH | PATHS

<any path search> ::=
  ANY [ <number of paths> ] [ <path mode> ] [ <path or paths> ]

<number of paths> ::=
  <simple value specification>

<shortest path search> ::=
    <all shortest path search>
  | <any shortest path search>
  | <counted shortest path search>
  | <counted shortest group search>

<all shortest path search> ::=
  ALL SHORTEST [ <path mode> ] [ <path or paths> ]

<any shortest path search> ::=
  ANY SHORTEST [ <path mode> ] [ <path or paths> ]

<counted shortest path search> ::=
  SHORTEST <number of paths> [ <path mode> ] [ <path or paths> ]

<counted shortest group search> ::=
  SHORTEST [ <number of groups> ] [ <path mode> ] [ <path or paths> ] { GROUP | GROUPS }

<number of groups> ::=
  <simple value specification>
```

Syntax Rules

1) If a <parenthesized path pattern expression> does not specify a <path mode prefix>, then WALK PATHS is implicit.

2) If a <path pattern prefix> *PPP* does not specify <all path search>, then:

   a) Case:

      i) If *PPP* does not simply contain a <path mode>, then let *PM* be WALK.

      ii) Otherwise, let *PM* be the <path mode> simply contained in *PPP*.

   b) Case:

      i) If *PPP* does not simply contain a <number of paths> or <number of groups>, then let *N* be an <unsigned integer> whose value is 1 (one).

      ii) Otherwise, let *N* be the <number of paths> or <number of groups> simply contained in *PPP*. The declared type of *N* shall be exact numeric with scale 0 (zero). If *N* is a <literal>, then the value of *N* shall be positive.

   c) Case:

      i) If *PPP* is an <any path search>, then *PPP* is equivalent to:

      ```
      ANY N PM PATHS
      ```

      ii) If *PPP* is a <shortest path search>, then

      Case:

         1) If *PPP* is <all shortest path search>, then *PPP* is equivalent to:

         ```
         SHORTEST 1 PM GROUP
         ```

         2) If *PPP* is <any shortest path search>, then *PPP* is equivalent to:

         ```
         SHORTEST 1 PM PATH
         ```

         3) If *PPP* is <counted shortest path search>, then *PPP* is equivalent to:

         ```
         SHORTEST N PM PATHS
         ```

         4) If *PPP* is <counted shortest group search>, then *PPP* is equivalent to:

         ```
         SHORTEST N PM GROUPS
         ```

3) A <path pattern prefix> that specifies a <path mode> other than WALK is *restrictive*. A <parenthesized path pattern expression> that immediately contains a restrictive <path mode prefix> is *restrictive*.

4) A <path search prefix> other than <all path search> is *selective*. A <path pattern> that simply contains a selective <path search prefix> is *selective*.

5) Let *PPPE* be a selective <path pattern>.

   a) An element variable exposed by *PPPE* is an *interior variable* of *PPPE*.

   b) A vertex variable *LVV* is the *left boundary variable* of *PPPE* if all of the following are true:

      i) *PPPE* exposes *LVV* as an unconditional singleton variable.

      ii) *LVV* is declared in the first implicit or explicit <vertex pattern> *LVP* contained in *PPPE*.

iii) *LVP* is not contained in a <path pattern union> or <path multiset alternation> that is contained in *PPPE*.

c) A vertex variable *RVV* is the *right boundary variable* of *PPPE* if the all of the following are true:

i) *PPPE* exposes *RVV* as an unconditional singleton variable.

ii) *RVV* is declared in the last implicit or explicit <vertex pattern> *RVP* contained in *PPPE*.

iii) *RVP* is not contained in a <path pattern union> or <path multiset alternation> that is contained in *PPPE*.

d) An element variable that is exposed by *PPPE* that is neither a left boundary variable of *PPPE* nor a right boundary variable of *PPPE* is a *strict interior variable* of *PPPE*.

6) An element variable that is not declared in a selective <path pattern> is an *exterior variable*.

7) A strict interior variable of one selective <path pattern> shall not be equivalent to an exterior variable, nor to an interior variable of another selective <path pattern>.

NOTE 127 — Implicit joins of boundary variables of selective <path pattern>s with exterior variables or boundary variables of other selective <path pattern>s are permitted.

8) A selective <path pattern> *SPP* shall not contain a reference to a graph pattern variable that is not declared by *SPP*.

NOTE 128 — This rule, and the prohibition of implicit joins to exterior variables and interior variables of other selective <path pattern>s, insure that each selective <path pattern> can be evaluated in isolation from any other <path pattern>.

Access Rules

None.

General Rules

None.

NOTE 129 — Restrictive <path mode>s are enforced as part of the check for consistent path bindings in the generation of the set of local matches in Subclause 10.6, “<path pattern expression>”. Selective <path pattern>s are evaluated by Subclause 9.8, “Evaluation of a selective <path pattern>”.

Conformance Rules

1) Without Feature G010, “Explicit WALK keyword”, conforming SQL language shall not contain a <path mode> that specifies WALK.

2) Without Feature G011, “Advanced path modes: TRAIL”, conforming SQL language shall not contain a <path mode> that specifies TRAIL.

3) Without Feature G012, “Advanced path modes: SIMPLE”, conforming SQL language shall not contain a <path mode> that specifies SIMPLE.

4) Without Feature G013, “Advanced path modes: ACYCLIC”, conforming SQL language shall not contain a <path mode> that specifies ACYCLIC.

5) Without Feature G014, “Explicit PATH/PATHS keywords”, conforming SQL language shall not contain a <path or paths>.

6) Without Feature G015, “All path search: explicit ALL keyword”, conforming SQL language shall not contain an <all path search>.

7) Without Feature G016, “Any path search”, conforming SQL language shall not contain an <any path search>.

8) Without Feature G017, “All shortest path search”, conforming SQL language shall not contain <all shortest path search>.

9) Without Feature G018, “Any shortest path search”, conforming SQL language shall not contain an <any shortest path search>.

10) Without Feature G019, “Counted shortest path search”, conforming SQL language shall not contain a <counted shortest path search>.

11) Without Feature G020, “Counted shortest group search”, conforming SQL language shall not contain a <counted shortest group search>.

10.6 <path pattern expression>

Function

Specify a pattern to match a single path in a property graph.

Format

```
<path pattern expression> ::=
    <path term>
  | <path multiset alternation>
  | <path pattern union>

<path multiset alternation> ::=
  <path term> <multiset alternation operator> <path term>
      [ { <multiset alternation operator> <path term> }... ]

<path pattern union> ::=
  <path term> <vertical bar> <path term> [ { <vertical bar> <path term> }... ]

<path term> ::=
    <path factor>
  | <path concatenation>

<path concatenation> ::=
  <path term> <path factor>

<path factor> ::=
    <path primary>
  | <quantified path primary>
  | <questioned path primary>

<quantified path primary> ::=
  <path primary> <graph pattern quantifier>

<questioned path primary> ::=
  <path primary> <question mark>
```

NOTE 130 — Unlike most regular expression languages, <question mark> is not equivalent to the quantifier {0,1}: the quantifier {0,1} exposes variables as group, whereas <question mark> does not change the singleton variables that it exposes to group. However, <question mark> does expose any singleton variables as conditional singletons.

```
<path primary> ::=
    <element pattern>
  | <parenthesized path pattern expression>
  | <simplified path pattern expression>

<element pattern> ::=
    <vertex pattern>
  | <edge pattern>

<vertex pattern> ::=
  <left paren> <element pattern filler> <right paren>

<element pattern filler> ::=
  [ <element variable declaration> ]
  [ <is label expression> ]
  [ <element pattern where clause> ]

<element variable declaration> ::=
  <element variable>
```

```
<is label expression> ::=
  <is or colon> <label expression>

<is or colon> ::=
    IS
  | <colon>

<element pattern where clause> ::=
  WHERE <search condition>

<edge pattern> ::=
    <full edge pattern>
  | <abbreviated edge pattern>

<full edge pattern> ::=
    <full edge pointing left>
  | <full edge undirected>
  | <full edge pointing right>
  | <full edge left or undirected>
  | <full edge undirected or right>
  | <full edge left or right>
  | <full edge any direction>

<full edge pointing left> ::=
  <left arrow bracket> <element pattern filler> <right bracket minus>

<full edge undirected> ::=
  <tilde left bracket> <element pattern filler> <right bracket tilde>

<full edge pointing right> ::=
  <minus left bracket> <element pattern filler> <bracket right arrow>

<full edge left or undirected> ::=
  <left arrow tilde bracket> <element pattern filler> <right bracket tilde>

<full edge undirected or right> ::=
  <tilde left bracket> <element pattern filler> <bracket tilde right arrow>

<full edge left or right> ::=
  <left arrow bracket> <element pattern filler> <bracket right arrow>

<full edge any direction> ::=
  <minus left bracket> <element pattern filler> <right bracket minus>

<abbreviated edge pattern> ::=
    <left arrow>
  | <tilde>
  | <right arrow>
  | <left arrow tilde>
  | <tilde right arrow>
  | <left minus right>
  | <minus sign>

<parenthesized path pattern expression> ::=
  <left paren>
      [ <subpath variable declaration> ]
      [ <path mode prefix> ]
      <path pattern expression>
      [ <parenthesized path pattern where clause> ]
  <right paren>

<subpath variable declaration> ::=
  <subpath variable> <equals operator>
```

```
<parenthesized path pattern where clause> ::=
  WHERE <search condition>
```

Syntax Rules

1) Let *RIGHTMINUS* be the following collection of <token>s: <right bracket minus>, <left arrow>, <slash minus>, and <minus sign>.

   NOTE 131 — These are the tokens]-, <-, /-, and -, which expose a minus sign on the right.

2) Let *LEFTMINUS* be the following collection of <token>s: <minus left bracket>, <right arrow>, <minus slash>, and <minus sign>.

   NOTE 132 — These are the tokens -[, ->, -/, and -, which expose a minus sign on the left. <minus sign> itself is in both *RIGHTMINUS* and *LEFTMINUS*.

3) A <path pattern expression> shall not juxtapose a <token> from *RIGHTMINUS* followed by a <token> from *LEFTMINUS* without a <separator> between them.

   NOTE 133 — Otherwise, the concatenation of the two tokens would include the sequence of two <minus sign>s, which is a <simple comment introducer>.

4) A <path pattern expression> that contains at the same depth of graph pattern matching a variable quantifier, a <questioned path primary>, a <path multiset alternation>, or a <path pattern union> is a *possibly variable length path pattern*.

5) A <path pattern expression> that is not a possibly variable length path pattern is a *fixed length path pattern*.

6) The *minimum path length* of certain BNF non-terminals defined in this Subclause is defined recursively as follows:

   a) The minimum path length of a <vertex pattern> is 0 (zero).

   b) The minimum path length of an <edge pattern> is 1 (one).

   c) The minimum path length of a <path concatenation> is the sum of the minimum path lengths of its operands.

   d) The minimum path length of a <path pattern union> or <path multiset alternation> is the minimum of the minimum path length of its operands.

   e) The minimum path length of a <quantified path primary> is the product of the minimum path length of the simply contained <path primary> and the value of the <lower bound>.

   f) The minimum path length of a <questioned path primary> is 0 (zero).

   g) The minimum path length of a <parenthesized path pattern expression> is the minimum path length of the simply contained <path pattern expression>.

   h) If *BNT1* and *BNT2* are two BNF non-terminals such that *BNT1* ::= *BNT2* and the minimum path length of *BNT2* is defined, then the minimum path length of *BNT1* is also defined and is the same as the minimum path length of *BNT2*.

7) The <path primary> immediately contained in a <quantified path primary> or <questioned path primary> shall have minimum path length that is greater than 0 (zero).

8) The <path primary> simply contained in a <quantified path primary> shall not contain a <quantified path primary> at the same depth of graph pattern matching.

9) Let *PMA* be a <path multiset alternation>.

   a) A <path term> simply contained in *PMA* is a *multiset alternation operand* of *PMA*.

b) Let *NOPMA* be the number of multiset alternation operands of *PMA*. Let $OPMA_1$, ..., $OPMA_{NOPMA}$ be an enumeration of the operands of *PMA*.

c) Any <subpath variable>s declared by <subpath variable declaration>s simply contained in the multiset alternation operands of *PMA* shall be mutually distinct.

d) Let $SOPMA_1$, ..., $SOPMA_{NOPMA}$ be implementation-dependent (UV024) <identifier>s that are mutually distinct and distinct from every <element variable>, <subpath variable> and <path variable> contained in *GP*.

e) For every *i*, 1 (one) ≤ *i* ≤ *NOPMA*,

Case:

i) If $OPMA_i$ is a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>, then let $OPMAX_i$ be $OPMA_i$.

ii) Otherwise, let $OPMAX_i$ be the <parenthesized path pattern expression>

```
(SOPMAi = OPMAi)
```

f) *PMA* is equivalent to:

```
OPMAX1 | ... | OPMAXNOPMA
```

10) A <path term> *PPUOP* simply contained in a <path pattern union> *PSD* is a *path pattern union operand* of *PSD*.

PPUOP shall not contain a reference to an element variable that is not declared in *PPUOP* or outside of *PSD*.

11) An <element pattern> *EP* that contains an <element pattern where clause> *EPWC* is transformed as follows:

a) Let *EPF* be the <element pattern filler> simply contained in *EP*.

b) Let *PREFIX* be the <delimiter token> contained in *EP* before *EPF* and let *SUFFIX* be the <delimiter token> contained in *EP* after *EPF*.

c) Let *EV* be the <element variable> simply contained in *EPF*. Let *ILE* be the <is label expression> contained in *EPF*, if any, otherwise, let *ILE* be the zero-length string.

d) *EP* is replaced by

```
( PREFIX EV ILE SUFFIX EPWC )
```

12) An <element pattern> that does not contain an <element variable declaration>, an <is label expression>, or an <element pattern where clause> is said to be *empty*.

13) Each <path pattern expression> is transformed in the following steps:

a) If the <path primary> immediately contained in a <quantified path primary> or <questioned path primary> is an <edge pattern> *EP*, then *EP* is replaced by

```
( EP )
```

NOTE 134 — For example,

```
->*
```

becomes:

```
(->) {0,}
```

which in later transformations becomes:

```
(() -> ()) {0,}
```

b) If two successive <element pattern>s contained in a <path concatenation> at the same depth of graph pattern matching are <edge pattern>s, then an implicit empty <vertex pattern> is inserted between them.

c) If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not preceded by a <vertex pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <vertex pattern> *VP* is inserted in *PST* immediately prior to *EP*.

d) If an edge pattern *EP* contained in a <path term> *PST* at the same depth of graph pattern matching is not followed by a <vertex pattern> contained in *PST* at the same depth of graph pattern matching, then an implicit empty <vertex pattern> *VP* is inserted in *PST* immediately after *EP*.

NOTE 135 — As a result of the preceding transformations, a fixed length path pattern has an odd number of <element pattern>s, beginning and ending with <vertex pattern>s, and alternating between <vertex pattern>s and <edge pattern>s.

e) Every <abbreviated edge pattern> *AEP* is replaced with an empty <full edge pattern> as follows.

Case:

i) If *AEP* is <left arrow>, then *AEP* is replaced by the <full edge pointing left>:

```
<-[ ]-
```

ii) If *AEP* is <tilde>, then *AEP* is replaced by the <full edge undirected>:

```
~[ ]~
```

iii) If *AEP* is <right arrow>, then *AEP* is replaced by the <full edge pointing right>:

```
-[ ]->
```

iv) If *AEP* is <left arrow tilde>, then *AEP* is replaced by the <full edge left or undirected>:

```
<~[ ]~
```

v) If *AEP* is <tilde right arrow>, then *AEP* is replaced by the <full edge undirected or right>:

```
~[ ]~>
```

vi) If *AEP* is <left minus right>, then *AEP* is replaced by the <full edge left or right>:

```
<-[ ]->
```

vii) If *AEP* is <minus sign>, then *AEP* is replaced by the <full edge any direction>:

```
-[ ]-
```

14) The *minimum vertex count* of certain BNF non-terminals defined in this Subclause is defined recursively as follows:

a) The minimum vertex count of a <vertex pattern> is 1 (one).

b) The minimum vertex count of an <edge pattern> is 0 (zero).

c) The minimum vertex count of a <path concatenation> *PC* is:

Case:

i) If two successive <element pattern>s contained in *PC* at the same depth of graph pattern matching are <vertex pattern>s, then 1 (one) less than the sum of the minimum vertex counts of its operands.

ii) Otherwise, the sum of the minimum vertex counts of its operands.

d) The minimum vertex count of a <path pattern union> or <path multiset alternation> is the minimum of the minimum vertex count of its operands.

e) The minimum vertex count of a <quantified path primary> is the product of the minimum vertex count of the simply contained <path primary> and the value of the <lower bound> of the simply contained <graph pattern quantifier>.

f) The minimum vertex count of a <questioned path primary> is 0 (zero).

g) The minimum vertex count of a <parenthesized path pattern expression> is the minimum vertex count of the simply contained <path pattern expression>.

h) If *BNF1* and *BNF2* are two BNF non-terminals such that *BNF1* ::= *BNF2* and the minimum vertex count of *BNF2* is defined, then the minimum vertex count of *BNF1* is also defined and is the same as the minimum vertex count of *BNF2*.

15) The <path pattern expression> simply contained in a <path pattern> shall have a minimum vertex count that is greater than 0 (zero).

NOTE 136 — The minimum vertex count must be computed after the syntactic transform that adds implicit vertex patterns. Thus a single <edge pattern> is a permitted <path pattern> because it implies two <vertex pattern>s.

NOTE 137 — A later Syntax Rule makes the same requirement of a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>.

16) An <element variable> *EV* contained in an <element variable declaration> *GPVD* is said to be *declared* by *GPVD*, and by the <element pattern> *EP* that simply contains *GPVD*. The <element variable> is the name of an element variable, which is also declared by *GPVD* and *EP*. *EV* is a *primary variable*.

17) An element variable that is declared by a <vertex pattern> is a *vertex variable*. An element variable that is declared by an <edge pattern> is an *edge variable*.

18) The scope of an <element variable> that is declared by an <element pattern> *EP* is the innermost <graph table> containing *EP*.

19) A <subpath variable> *SV* contained in a <subpath variable declaration> *SVD* is said to be *declared* by *SVD*, and by the <parenthesized path pattern expression> *PPPE* that simply contains *SVD*. *SV* is the name of a subpath variable, which is also declared by *SVD* and *PPPE*. If *PPPE* simply contains a <subpath variable declaration>, then the minimum vertex count of *PPPE* shall be greater than 0 (zero).

20) If *EP* is an <element pattern> that contains an <element pattern where clause> *EPWC*, then *EP* shall simply contain an <element variable declaration> *GPVD*.

21) If *EV* is an element variable or subpath variable, and *BNT* is an instance of a BNF non-terminal, then the terminology "*BNT* exposes *EV*" is defined as follows. The full terminology is one of the following: "*BNT* exposes *EV* as an unconditional singleton variable", "*BNT* exposes *EV* as a conditional singleton variable", "*BNT* exposes *EV* as an effectively bounded group variable", or "*BNT* exposes *EV* as an effectively unbounded group variable". The terms "unconditional singleton variable", "conditional singleton variable", "effectively bounded group variable", and "effectively unbounded group variable" are called the *degree of exposure*.

a) An <element pattern> *EP* that declares an element variable *EV* exposes *EV* as an unconditional singleton.

b) A <parenthesized path pattern expression> *PPPE* that simply contains a <subpath variable declaration> that declares *EV* exposes *EV* as an unconditional singleton variable. *PPPE* shall not contain another <parenthesized path pattern expression> that declares *EV*.

c) If a <path concatenation> *PPC* declares *EV*, then let *PT* be the <path term> and let *PF* be the <path factor> simply contained in *PPC*.

Case:

i) If *EV* is exposed as an unconditional singleton by both *PT* and *PF*, then *EV* is exposed as an unconditional singleton by *PPC*. *EV* shall not be a subpath variable.

NOTE 138 — This case expresses an implicit join on *EV* within *PPC*. Implicit joins between conditional singleton variables, group variables, or subpath variables are forbidden.

ii) Otherwise, *EV* shall only be exposed by one of *PT* or *PF*. In this case *EV* is exposed by *PPC* in the same degree that it is exposed by *PT* or *PF*.

d) If a <path pattern union> or <path multiset alternation> *PA* declares *EV*, then

Case:

i) If every operand of *PA* exposes *EV* as an unconditional singleton variable, then *PA* exposes *EV* as an unconditional singleton variable.

ii) If at least one operand of *PA* exposes *EV* as an effectively unbounded group variable, then *PA* exposes *EV* as an effectively unbounded group variable.

iii) If at least one operand of *PA* exposes *EV* as an effectively bounded group variable, then *PA* exposes *EV* as an effectively bounded group variable.

iv) Otherwise, *PA* exposes *EV* as a conditional singleton variable.

e) If a <quantified path primary> *QPP* declares *EV*, then let *PP* be the <path primary> simply contained in *QPP*.

Case:

i) If *QPP* contains a <graph pattern quantifier> that is a <fixed quantifier> or a <general quantifier> that contains an <upper bound> and *PP* does not expose *EV* as an effectively unbounded group variable, then *QPP* exposes *EV* as an effectively bounded group variable.

ii) If *QPP* is contained at the same depth of graph pattern matching in a restrictive <parenthesized path pattern expression>, then *QPP* exposes *EV* as an effectively bounded group variable.

NOTE 139 — The preceding definition must be applied after the syntactic transformation to insure that every <path mode prefix> is at the head of a <parenthesized path pattern expression>.

iii) Otherwise, *QPP* exposes *EV* as an effectively unbounded group variable.

f) If a <questioned path primary> *QUPP* declares *EV*, then let *PP* be the <path primary> simply contained in *QUPP*.

Case:

i) If *PP* exposes *EV* as a group variable, then *QUPP* exposes *EV* as a group variable with the same degree of exposure.

ii) Otherwise, *QUPP* exposes *EV* as a conditional singleton variable.

g) A <parenthesized path pattern expression> exposes the same variables as the simply contained <path pattern expression>, in the same degree of exposure.

NOTE 140 — A restrictive <path mode> declared by a <parenthesized path pattern expression> makes variables effectively bounded, but it does so even for proper subexpressions within the scope of the <path mode> and has already been handled by the rules for <quantified path primary>.

h) If a <path pattern> *PP* declares *EV*, then let *PPE* be the simply contained <path pattern expression>.

   Case:

   i) If *PPE* exposes *EV* as an unconditional singleton, a conditional singleton, or an effectively bounded group variable, then *PP* exposes *EV* with the same degree of exposure.

   ii) Otherwise, *PP* exposes *EV* as an effectively bounded group variable.

   NOTE 141 — That is, even if *PPE* exposes *EV* as an effectively unbounded group variable, *PP* still exposes *EV* as effectively bounded, because in this case *PP* must be a selective <path pattern>.

i) If *BNT1* and *BNT2* are two BNF non-terminals such that *BNT1* ::= *BNT2* and *BNT2* exposes *EV*, then *BNT1* exposes *EV* to the same degree of exposure as *BNT2*.

22) If *BNT* is a BNF non-terminal that exposes a graph pattern variable *GPV* with a degree of exposure *DEGREE*, then *BNT* is also said to expose the name of *GPV* with degree of exposure *DEGREE*.

23) A <parenthesized path pattern where clause> *PPPWC* simply contained in a <parenthesized path pattern expression> *PPPE* shall not reference a path variable.

Access Rules

None.

General Rules

None.

NOTE 142 — The evaluation of a <path pattern expression> is performed by the General Rules of Subclause 9.7, "Evaluation of a <path pattern expression>".

Conformance Rules

1) Without Feature G030, "Path multiset alternation", conforming SQL language shall not contain a <path multiset alternation>.

2) Without Feature G031, "Path multiset alternation: variable length path operands", in conforming SQL language, an operand of a <path multiset alternation> shall be a fixed length path pattern.

3) Without Feature G032, "Path pattern union", conforming SQL language shall not contain a <path pattern union>.

4) Without Feature G033, "Path pattern union: variable length path operands", in conforming SQL language, an operand of a <path pattern union> shall be a fixed length path pattern.

5) Without Feature G034, "Path concatenation", conforming SQL language shall not contain a <path concatenation>.

6) Without Feature G035, "Quantified paths", conforming SQL language shall not contain a <quantified path primary> that does not immediately contain a <path primary> that is an <edge pattern>.

7) Without Feature G036, "Quantified edges", conforming SQL language shall not contain a <quantified path primary> that immediately contains a <path primary> that is an <edge pattern>.

8) Without Feature G037, “Questioned paths”, conforming SQL language shall not contain a <questioned path primary>.

9) Without Feature G038, “Parenthesized path pattern expression”, conforming SQL language shall not contain a <parenthesized path pattern expression>.

10) Without Feature G040, “Vertex pattern”, conforming SQL language shall not contain a <vertex pattern>.

11) Without Feature G041, “Non-local element pattern predicates”, in conforming SQL language, the <element pattern where clause> of an <element pattern> *EP* shall only reference the <element variable> declared in *EP*.

12) Without Feature G042, “Basic full edge patterns”, conforming SQL language shall not contain a <full edge any direction>, a <full edge pointing left>, or a <full edge pointing right>.

13) Without Feature G043, “Complete full edge patterns”, conforming SQL language shall not contain a <full edge pattern> that is not a <full edge any direction>, a <full edge pointing left>, or a <full edge pointing right>.

14) Without Feature G044, “Basic abbreviated edge patterns”, conforming SQL language shall not contain an <abbreviated edge pattern> that is a <minus sign>, <left arrow>, or <right arrow>.

15) Without Feature G045, “Complete abbreviated edge patterns”, conforming SQL language shall not contain an <abbreviated edge pattern> that is not a <minus sign>, <left arrow>, or <right arrow>.

16) Without Feature G046, “Relaxed topological consistency: adjacent vertex patterns”, in conforming SQL language, between any two <vertex pattern>s contained in a <path pattern expression> there shall be at least one <edge pattern>, <left paren>, or <right paren>.

17) Without Feature G047, “Relaxed topological consistency: concise edge patterns”, in conforming SQL language, any <edge pattern> shall be immediately preceded and followed by a <vertex pattern>.

18) Without Feature G048, “Parenthesized path pattern: subpath variable declaration”, conforming SQL language shall not contain a <parenthesized path pattern expression> that simply contains a <subpath variable declaration>.

19) Without Feature G049, “Parenthesized path pattern: path mode prefix”, conforming SQL language shall not contain a <parenthesized path pattern expression> that immediately contains a <path mode prefix>.

20) Without Feature G050, “Parenthesized path pattern: WHERE clause”, conforming SQL language shall not contain a <parenthesized path pattern where clause>.

21) Without Feature G051, “Parenthesized path pattern: non-local predicates”, in conforming SQL language, a <parenthesized path pattern where clause> simply contained in a <parenthesized path pattern expression> *PPPE* shall not reference an <element variable> that is not declared in *PPPE*.

22) Without Feature G830, “Colon in ‘is label’ expression”, conforming SQL language shall not contain an <is or colon> that is a <colon>.

10.7 <graph pattern quantifier>

Function

Specify a graph pattern quantifier.

Format

```
<graph pattern quantifier> ::=
    <asterisk>
  | <plus sign>
  | <fixed quantifier>
  | <general quantifier>

<fixed quantifier> ::=
  <left brace> <unsigned integer> <right brace>

<general quantifier> ::=
  <left brace> [ <lower bound> ] <comma> [ <upper bound> ] <right brace>

<lower bound> ::=
  <unsigned integer>

<upper bound> ::=
  <unsigned integer>
```

Syntax Rules

1) The maximum value of <upper bound> is implementation-defined (IL012). <upper bound>, if specified, shall not be greater than this value.

2) Every <graph pattern quantifier> is normalized, as follows:

   a) <asterisk> is equivalent to:

      `{0,}`

   b) <plus sign> is equivalent to:

      `{1,}`

   c) If <fixed quantifier> *FQ* is specified, then let *UI* be the <unsigned integer> contained in *FQ*. *FQ* is equivalent to:

      `{`*`UI`*`,`*`UI`*`}`

   d) If <general quantifier> *GQ* is specified, and if <lower bound> is not specified, then the <unsigned integer> 0 (zero) is supplied as the <lower bound>.

3) If <general quantifier> *GQ* is specified or implied by the preceding normalizations, then:

   Case:

   a) If <upper bound> is specified, then:

      i) The value of <upper bound> *VUP* shall be greater than 0 (zero).

      ii) The value of <lower bound> *LUP* shall be less than or equal to *VUP*.

iii) If *LUP* equals *VUP*, then *GQ* is a *fixed quantifier*.

iv) *GQ* is a *bounded quantifier*.

b) Otherwise, *GQ* is an *unbounded quantifier*.

4) A <graph pattern quantifier> that is not a fixed quantifier is a *variable quantifier*.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature G060, “Bounded graph pattern quantifiers”, conforming SQL language shall not contain a <fixed quantifier> or a <general quantifier> that immediately contains an <upper bound>.

2) Without Feature G061, “Unbounded graph pattern quantifiers”, conforming SQL language shall not contain a <graph pattern quantifier> that immediately contains <asterisk>, <plus sign>, or a <general quantifier> that does not immediately contain an <upper bound>.

10.8 <label expression>

Function

Specify an expression that matches one or more labels of a property graph.

Format

```
<label expression> ::=
    <label term>
  | <label disjunction>

<label disjunction> ::=
  <label expression> <vertical bar> <label term>

<label term> ::=
    <label factor>
  | <label conjunction>

<label conjunction> ::=
  <label term> <ampersand> <label factor>

<label factor> ::=
    <label primary>
  | <label negation>

<label negation> ::=
  <exclamation mark> <label primary>

<label primary> ::=
    <label name>
  | <wildcard label>
  | <parenthesized label expression>

<wildcard label> ::=
  <percent>

<parenthesized label expression> ::=
  <left paren> <label expression> <right paren>
```

Syntax Rules

1) Let *LE* be the <label expression>.

2) Let *GP* be the <graph pattern> that simply contains *LE*. Let *GT* be the <graph table> that simply contains *GP*. Let *GR* be the <graph reference> that is simply contained in *GT*. Let *PG* be the property graph that is identified by *GR*.

3) Every <label name> contained in *LE* shall identify a label of *PG*.

4) Case:

   a) If *LE* is simply contained in a <vertex pattern>, then *LE* is a *vertex <label expression>*. Every <label name> contained in *LE* shall identify a vertex label of *PG*.

   b) Otherwise, *LE* is an *edge <label expression>*. Every <label name> contained in *LE* shall identify an edge label of *PG*.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature G070, “Label expression: label disjunction”, conforming SQL language shall not contain a <label disjunction>.

2) Without Feature G071, “Label expression: label conjunction”, conforming SQL language shall not contain a <label conjunction>.

3) Without Feature G072, “Label expression: label negation”, conforming SQL language shall not contain a <label negation>.

4) Without Feature G073, “Label expression: individual label name”, conforming SQL language shall not contain a <label expression> that is a <label name>.

5) Without Feature G074, “Label expression: wildcard label”, conforming SQL language shall not contain a <wildcard label>.

6) Without Feature G075, “Parenthesized label expression”, conforming SQL language shall not contain a <parenthesized label expression>.

10.9 <simplified path pattern expression>

Function

Express a path pattern as a regular expression of edge labels.

Format

```
<simplified path pattern expression> ::=
    <simplified defaulting left>
  | <simplified defaulting undirected>
  | <simplified defaulting right>
  | <simplified defaulting left or undirected>
  | <simplified defaulting undirected or right>
  | <simplified defaulting left or right>
  | <simplified defaulting any direction>

<simplified defaulting left> ::=
  <left minus slash> <simplified contents> <slash minus>

<simplified defaulting undirected> ::=
  <tilde slash> <simplified contents> <slash tilde>

<simplified defaulting right> ::=
  <minus slash> <simplified contents> <slash minus right>

<simplified defaulting left or undirected> ::=
  <left tilde slash> <simplified contents> <slash tilde>

<simplified defaulting undirected or right> ::=
  <tilde slash> <simplified contents> <slash tilde right>

<simplified defaulting left or right> ::=
  <left minus slash> <simplified contents> <slash minus right>

<simplified defaulting any direction> ::=
  <minus slash> <simplified contents> <slash minus>

<simplified contents> ::=
    <simplified term>
  | <simplified path union>
  | <simplified multiset alternation>

<simplified path union> ::=
  <simplified term> <vertical bar> <simplified term>
      [ { <vertical bar> <simplified term> }... ]

<simplified multiset alternation> ::=
  <simplified term> <multiset alternation operator> <simplified term>
      [ { <multiset alternation operator> <simplified term> }... ]

<simplified term> ::=
    <simplified factor low>
  | <simplified concatenation>

<simplified concatenation> ::=
  <simplified term> <simplified factor low>

<simplified factor low> ::=
    <simplified factor high>
  | <simplified conjunction>
```

```
<simplified conjunction> ::=
  <simplified factor low> <ampersand> <simplified factor high>

<simplified factor high> ::=
    <simplified tertiary>
  | <simplified quantified>
  | <simplified questioned>

<simplified quantified> ::=
  <simplified tertiary> <graph pattern quantifier>

<simplified questioned> ::=
  <simplified tertiary> <question mark>

<simplified tertiary> ::=
    <simplified direction override>
  | <simplified secondary>

<simplified direction override> ::=
    <simplified override left>
  | <simplified override undirected>
  | <simplified override right>
  | <simplified override left or undirected>
  | <simplified override undirected or right>
  | <simplified override left or right>
  | <simplified override any direction>

<simplified override left> ::=
  <left angle bracket> <simplified secondary>

<simplified override undirected> ::=
  <tilde> <simplified secondary>

<simplified override right> ::=
  <simplified secondary> <right angle bracket>

<simplified override left or undirected> ::=
  <left arrow tilde> <simplified secondary>

<simplified override undirected or right> ::=
  <tilde> <simplified secondary> <right angle bracket>

<simplified override left or right> ::=
  <left angle bracket> <simplified secondary> <right angle bracket>

<simplified override any direction> ::=
  <minus sign> <simplified secondary>

<simplified secondary> ::=
    <simplified primary>
  | <simplified negation>

<simplified negation> ::=
  <exclamation mark> <simplified primary>

<simplified primary> ::=
    <label name>
  | <left paren> <simplified contents> <right paren>
```

Syntax Rules

1) A <simplified negation> shall not contain a <simplified concatenation>, <simplified quantified>, <simplified questioned>, or <simplified multiset alternation>.

2) A <simplified direction override> shall not contain another <simplified direction override>.

3) A <simplified direction override> shall not contain <simplified concatenation>, <simplified quantified>, <simplified questioned>, or <simplified multiset alternation>.

4) A <simplified conjunction> shall not contain a <simplified concatenation>, <simplified quantified>, <simplified questioned>, or <simplified multiset alternation>.

5) A <simplified path pattern expression> *SPPE* is replaced by:

```
( SPPE )
```

NOTE 143 — This is done once for each <simplified path pattern expression> prior to the following recursive transformation and not with each iteration of the transformation.

6) The following rules are recursively applied until no <simplified path pattern expression>s remain.

NOTE 144 — The rules work from the root of the parse tree of a <simplified path pattern expression>. At each step, the coarsest analysis of a <simplified path pattern expression> is replaced, eliminating at least one level of the parse tree, measured from the root. Note that each replacement can create more <simplified path pattern expression>s than before, but these replacements have less depth. Eventually the recursion replaces <simplified path pattern expression> with <edge pattern>.

a) Let *SPPE* be a <simplified path pattern expression>.

i) Let *SC* be the <simplified contents> contained in *SPPE*.

ii) Let *PREFIX* be the <minus slash>, <left minus slash>, <tilde slash>, <left tilde slash>, or <left minus slash> contained in *SPPE*.

iii) Let *SUFFIX* be the <slash minus right>, <slash minus>, <slash tilde>, or <slash tilde right> contained in *SPPE*.

iv) Let *EDGEPRE* and *EDGESUF* be determined by Table 1, "Conversion of simplified syntax delimiters to default edge delimiters", from the row containing the values of *PREFIX* and *SUFFIX*.

Table 1 — Conversion of simplified syntax delimiters to default edge delimiters

PREFIX	*SUFFIX*	*EDGEPRE*	*EDGESUF*
`-/`	`/->`	`-[`	`]->`
`<-/`	`/-`	`<-[`	`]-`
`~/`	`/~`	`~[`	`]~`
`~/`	`/~>`	`~[`	`]~>`
`<~/`	`/~`	`<~[`	`]~`
`<-/`	`/->`	`<-[`	`]->`
`-/`	`/-`	`-[`	`]-`

b) Case:

i) If *SC* is a <simplified path union> *SPU*, then let *N* be the number of <simplified term>s simply contained in *SPU*, and let ST_1, ..., ST_N be those <simplified term>s; *SPPE* is replaced by:

```
PREFIX ST1 SUFFIX | PREFIX ST2 SUFFIX | ... | PREFIX STN SUFFIX
```

ii) If *SC* is a <simplified multiset alternation> *SMA*, then let *N* be the number of <simplified term>s simply contained in *SMA*, and let ST_1, ..., ST_N be those <simplified term>s; *SPPE* is replaced by:

```
PREFIX ST1 SUFFIX |+| PREFIX ST2 SUFFIX |+| ... |+| PREFIX STN SUFFIX
```

iii) If *SC* is a <simplified concatenation> *SCAT*, then let *ST* be the <simplified term> and let *SFL* be the <simplified factor low> simply contained in *SCAT*; *SPPE* is replaced by:

```
PREFIX ST SUFFIX PREFIX SFL SUFFIX
```

iv) If *SC* is a <simplified conjunction> *SAND*, then *SPPE* is replaced by:

```
EDGEPRE IS SAND EDGESUF
```

NOTE 145 — As a result, *SAND* is now interpreted as a <label expression> within an <edge pattern>. By earlier Syntax Rules, there are no operators allowed in *SAND* that cannot be interpreted as operators of a <label expression>.

v) If *SC* is a <simplified quantified> *SQ*, then let *ST* be the <simplified tertiary> simply contained in *SC* and let *GPQ* be the <graph pattern quantifier> simply contained in *SQ*; *SPPE* is replaced by:

```
( PREFIX ST SUFFIX ) GPQ
```

vi) If *SC* is a <simplified questioned> *SQU*, then let *ST* be the <simplified tertiary> simply contained in *SC*; *SPPE* is replaced by:

```
( PREFIX ST SUFFIX ) ?
```

vii) If *SC* is a <simplified direction override> *SDO*, then let *SS* be the <simplified secondary> simply contained in *SDO*.

Case:

NOTE 146 — As a result of the following replacements, *SDO* is now interpreted as a <label expression> within an <edge pattern>. By earlier Syntax Rules, there are no operators allowed in *SDO* that cannot be interpreted as operators of a <label expression>.

1) If *SDO* is <simplified override left>, then *SPPE* is replaced by:

```
<-[ IS SS ]-
```

2) If *SDO* is <simplified override undirected>, then *SPPE* is replaced by:

```
~[ IS SS ]~
```

3) If *SDO* is <simplified override left or undirected>, then *SPPE* is replaced by:

```
<~[ IS SS ]~
```

4) If *SDO* is <simplified override undirected or right>, then *SPPE* is replaced by:

```
~[ IS SS ]~>
```

5) If *SDO* is <simplified override left or right>, then *SPPE* is replaced by:

```
<-[ IS SS ]->
```

6) If *SDO* is <simplified override any direction>, then *SPPE* is replaced by:

```
-[ IS SS ]-
```

viii) If *SC* is a <simplified negation> *SN*, then *SPPE* is replaced by:

```
EDGEPRE IS SN EDGESUF
```

NOTE 147 — As a result, *SN* is now interpreted as a <label expression> within an <edge pattern>. By earlier Syntax Rules, there are no operators allowed in *SN* that cannot be interpreted as operators of a <label expression>.

ix) If *SC* is a <simplified primary> *SP*, then

Case:

1) If *SP* is a <label name>, then *SPPE* is replaced by:

```
EDGEPRE IS SP EDGESUF
```

2) Otherwise, let *INNER* be the <simplified contents> simply contained in *SC*; *SPPE* is replaced by:

```
( PREFIX INNER SUFFIX )
```

7) The Conformance Rules of Subclause 10.6, "<path pattern expression>" are applied to the result of the previous syntactic transformation.

Access Rules

None.

General Rules

None.

Conformance Rules

1) Without Feature G039, "Simplified path pattern expression: full defaulting", conforming SQL language shall not contain a <simplified path pattern expression> that is not a <simplified defaulting left>, a <simplified defaulting right>, or a <simplified defaulting any direction>.

2) Without Feature G080, "Simplified path pattern expression: basic defaulting", conforming SQL language shall not contain a <simplified defaulting left>, a <simplified defaulting right>, or a <simplified defaulting any direction>.

3) Without Feature G081, "Simplified path pattern expression: full overrides", conforming SQL language shall not contain a <simplified direction override> that is not a <simplified override left>, <simplified override right>, or a <simplified override any direction>.

4) Without Feature G082, "Simplified path pattern expression: basic overrides", conforming SQL language shall not contain a <simplified override left>, a <simplified override right>, or a <simplified override any direction>.

10.10 <element reference>

Function

Reference a graph element or list of graph elements in a property graph.

Format

```
<element reference> ::=
  <element variable>
```

Syntax Rules

1) Let *EV* be the <element variable> simply contained in the <element reference> *ER*. Let *GT* be the innermost <graph table> that declares *EV* and that contains *ER*. Let *GP* be the <graph pattern> simply contained in *GT*.

2) Let *V* be the element variable that is identified by *EV*. *ER references V*.

3) The *degree of reference* of *ER* is defined as follows.

   Case:

   a) If *ER* is simply contained in the <graph table columns clause> or <graph pattern where clause> of *GT*, then the degree of reference of *ER* is the degree of exposure of *EV* by *GP*.

      NOTE 148 — "Degree of exposure" is defined in Subclause 10.4, "<graph pattern>" and Subclause 10.6, "<path pattern expression>".

   b) If *ER* is contained in a <parenthesized path pattern where clause> *PPPWC* simply contained in *GP*, then let *PPPE* be the <parenthesized path pattern expression> that simply contains *PPPWC*.

      NOTE 149 — This rule is applied after the syntactic transform that converts any <element pattern where clause> to a <parenthesized path pattern where clause>.

      Case:

      i) If *EV* is declared by *PPPE*, then the degree of reference of *ER* is the degree of exposure of *EV* by *PPPE*.

         NOTE 150 — "Degree of exposure" is defined in Subclause 10.4, "<graph pattern>" and Subclause 10.6, "<path pattern expression>".

      ii) Otherwise, let *PP* be the innermost <graph pattern> or <parenthesized path pattern expression> that contains *PPPWC* and that declares *EV*. The degree of reference of *ER* is the degree of exposure of *EV* by *PP*. The degree of reference of *ER* shall be singleton.

         NOTE 151 — "Degree of exposure" is defined in Subclause 10.4, "<graph pattern>" and Subclause 10.6, "<path pattern expression>".

4) If the degree of reference of *ER* is group, then it shall be effectively bounded group, and *ER* shall be contained in an aggregated argument of a <set function specification>.

Access Rules

None.

General Rules

None.

NOTE 152 — Every <element reference> is evaluated in the General Rules of Subclause 9.9, “Applying bindings to evaluate an expression”.

Conformance Rules

None.

10.11 <path reference>

Function

Reference a path variable.

Format

```
<path reference> ::=
  <path variable>
```

Syntax Rules

1) The degree of reference of a <path reference> shall be unconditional singleton.

Access Rules

None.

General Rules

1) Let *PV* be the <path variable> simply contained in *GPLF*.

2) Let *PB* be the path binding of the current match corresponding to the <path pattern> that declares *PV*.

   NOTE 153 — The current match is specified in the General Rules of Subclause 7.1, "<table reference>".

3) *PB* is the value of the <path reference>.

Conformance Rules

None.

11 Schema definition and manipulation

This Clause modifies Clause 11, "Schema definition and manipulation", in ISO/IEC 9075-2.

11.1 <schema definition>

This Subclause modifies Subclause 11.1, "<schema definition>", in ISO/IEC 9075-2.

Function

Define a schema.

Format

```
<schema element> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <property graph definition>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.2 <drop schema statement>

This Subclause modifies Subclause 11.2, "<drop schema statement>", in ISO/IEC 9075-2.

Function

Destroy a schema.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)n): SQL-property graphs.

Access Rules

No additional Access Rules.

General Rules

1) Insert before GR 12): For every SQL-property graph included in *S*:
   a) Let *SPG* be the SQL-property graph and let *PG* be the <property graph name> included in the descriptor of *SPG*.
   b) The following <drop property graph statement> statement is effectively executed:

```
DROP PROPERTY GRAPH PG CASCADE
```

Conformance Rules

No additional Conformance Rules.

11.3 <table definition>

This Subclause modifies Subclause 11.3, “<table definition>”, in ISO/IEC 9075-2.

Function

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

Format

No additional Format items.

Syntax Rules

1) Insert after SR 7): *TN* shall not identify an existing SQL-property graph descriptor.

NOTE 154 — SQL-property graphs share the namespace with tables.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.4 <drop column definition>

This Subclause modifies Subclause 11.23, “<drop column definition>”, in ISO/IEC 9075-2.

Function

Destroy a column of a base table.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 7)e): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.5 <drop table statement>

This Subclause modifies Subclause 11.31, “<drop table statement>”, in ISO/IEC 9075-2.

Function

Destroy a table.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 7)g): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.6 <view definition>

This Subclause modifies Subclause 11.32, "<view definition>", in ISO/IEC 9075-2.

Function

Define a viewed table.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4): The schema identified by the explicit or implicit <schema name> of the <table name> shall not include an SQL-property graph descriptor whose SQL-property graph name is <table name>.

   NOTE 155 — SQL-property graphs share the namespace with tables.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.7 <drop view statement>

This Subclause modifies Subclause 11.33, “<drop view statement>”, in ISO/IEC 9075-2.

Function

Destroy a view.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 5)d): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.8 <drop domain statement>

This Subclause modifies Subclause 11.40, "<drop domain statement>", in ISO/IEC 9075-2.

Function

Destroy a domain.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 2)d): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.9 <drop character set statement>

This Subclause modifies Subclause 11.42, "<drop character set statement>", in ISO/IEC 9075-2.

Function

Destroy a character set.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)j): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.10 <drop collation statement>

This Subclause modifies Subclause 11.44, "<drop collation statement>", in ISO/IEC 9075-2.

Function

Destroy a collation.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)f): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.11 <drop transliteration statement>

This Subclause modifies Subclause 11.46, “<drop transliteration statement>”, in ISO/IEC 9075-2.

Function

Destroy a character transliteration.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)f): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.12 <drop attribute definition>

This Subclause modifies Subclause 11.55, “<drop attribute definition>”, in ISO/IEC 9075-2.

Function

Destroy an attribute of a user-defined type.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 8)a)iv): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.13 <drop data type statement>

This Subclause modifies Subclause 11.59, “<drop data type statement>”, in ISO/IEC 9075-2.

Function

Destroy a user-defined type.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)f)v): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

2) Insert after SR 4)h)i)4): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.14 <alter routine statement>

This Subclause modifies Subclause 11.61, “<alter routine statement>”, in ISO/IEC 9075-2.

Function

Alter a characteristic of an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)b)iv): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.15 <drop routine statement>

This Subclause modifies Subclause 11.62, "<drop routine statement>", in ISO/IEC 9075-2.

Function

Destroy an SQL-invoked routine.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)c)v): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.16 <drop user-defined cast statement>

This Subclause modifies Subclause 11.64, “<drop user-defined cast statement>”, in ISO/IEC 9075-2.

Function

Destroy a user-defined cast.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 7)d): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.17 <drop user-defined ordering statement>

This Subclause modifies Subclause 11.66, "<drop user-defined ordering statement>", in ISO/IEC 9075-2.

Function

Destroy a user-defined ordering method.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 4)d): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.18 <drop sequence generator statement>

This Subclause modifies Subclause 11.74, "<drop sequence generator statement>", in ISO/IEC 9075-2.

Function

Destroy an external sequence generator.

Format

No additional Format items.

Syntax Rules

1) Insert after SR 3)b): Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

11.19 <property graph definition>

Function

Define an SQL-property graph.

Format

```
<property graph definition> ::=
  CREATE PROPERTY GRAPH <property graph name> [ <property graph content> ]

<property graph content> ::=
  <vertex tables clause> [ <edge tables clause> ]

<vertex tables clause> ::=
  <vertex synonym> TABLES <parenthesized vertex table list>

<parenthesized vertex table list> ::=
  <left paren> <vertex table definition> [ { <comma> <vertex table definition> }... ]
      <right paren>

<vertex table definition> ::=
  <element table definition>

<element table definition> ::=
    <element table name> [ AS <element table alias> ]
        [ <element table key clause> ]
        [ <source vertex table> <destination vertex table> ]
        [ <element table label and properties clause> ]
  | <table subquery> AS <element table alias>
        <element table key clause>
        [ <source vertex table> <destination vertex table> ]
        [ <element table label and properties clause> ]

<element table key clause> ::=
  KEY <left paren> <column name list> <right paren>

<element table label and properties clause> ::=
    <element table properties clause>
  | <label and properties>...

<element table properties clause> ::=
    NO PROPERTIES
  | PROPERTIES <element table properties alternatives>

<element table properties alternatives> ::=
    [ ARE ] ALL COLUMNS [ EXCEPT <left paren> <except column name list> <right paren> ]
  | <element table parenthesized derived property list>

<except column name list> ::=
  <column name list>

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

<derived property> ::=
  <value expression> [ AS <property name> ]

<label and properties> ::=
  <element table label clause> [ <element table properties clause> ]
```

```
<element table label clause> ::=
    LABEL <label name>
  | DEFAULT LABEL

<edge tables clause> ::=
  <edge synonym> TABLES <parenthesized edge table list>

<parenthesized edge table list> ::=
  <left paren> <edge table definition>
      [ { <comma> <edge table definition> }... ] <right paren>

<edge table definition> ::=
  <element table definition>

<source vertex table> ::=
  SOURCE [ <source vertex table key clause> REFERENCES ] <source vertex reference>

<source vertex table key clause> ::=
  <element table key clause>

<source vertex reference> ::=
  <source vertex table alias>
      [ <left paren> <referenced source column list> <right paren> ]

<referenced source column list> ::=
  <column name list>

<destination vertex table> ::=
  DESTINATION [ <destination vertex table key clause> REFERENCES ]
      <destination vertex reference>

<destination vertex table key clause> ::=
  <element table key clause>

<destination vertex reference> ::=
  <destination vertex table alias>
      [ <left paren> <referenced destination column list> <right paren> ]

<referenced destination column list> ::=
  <column name list>
```

Syntax Rules

1) The <property graph content> shall not contain a <host parameter specification>, an <SQL parameter reference>, a <dynamic parameter specification>, an <embedded variable specification>, or an <SQL variable reference>.

   NOTE 156 — <SQL variable reference> is defined in ISO/IEC 9075-4.

2) Let *SPG* be the SQL-property graph defined by the <property graph definition> *PD*. Let *SPN* be the <property graph name> simply contained in *PD*.

3) If *PD* is contained in a <schema definition> *SD* and *SPN* contains a <schema name>, then that <schema name> shall be equivalent to the implicit or explicit <schema name> of *SD*.

4) *SPN* shall not identify an existing SQL-property graph descriptor or an existing table descriptor.

   NOTE 157 — SQL-property graphs share the namespace with tables.

5) If <vertex tables clause> *VTC* is specified, then let *n* be the number of <vertex table definition>s simply contained in *VTC*. For *i*, 1 (one) ≤ *i* ≤ *n*:

a) Let *VTD$_i$* be the *i*-th <vertex table definition> simply contained in *VTC*. *VTD$_i$* shall not simply contain <source vertex table> or <destination vertex table>.

b) The Syntax Rules of Subclause 9.13, "Creation of a vertex table descriptor", are applied with *VTD$_i$* as *BNFTERM*; let *VDESC$_i$* be the *DESCRIPTOR* returned from the application of those Syntax Rules.

c) *VDESC$_i$* is a vertex table descriptor and describes a vertex table of *SPG*.

6) If <edge tables clause> *ETC* is specified, then let *m* be the number of <edge table definition>s simply contained in *ETC*. For *i*, 1 (one) ≤ *i* ≤ *m*:

a) Let *ETD$_i$* be the *i*-th <edge table definition> simply contained in *ETC*. *ETD$_i$* shall simply contain a <source vertex table> *SVT$_i$* and a <destination vertex table> *DVT$_i$*.

b) The <source vertex table alias> simply contained in *SVT$_i$* shall be equivalent to the element table alias included in a unique vertex table descriptor *SVTD$_i$* instantiated in SR 5).

c) The <destination vertex table alias> simply contained in *DVT$_i$* shall be equivalent to the element table alias included in a unique vertex table descriptor *DVTD$_i$* instantiated in SR 5).

d) The Syntax Rules of Subclause 9.14, "Creation of an edge table descriptor", are applied with *ETD$_i$* as *BNFTERM*, *SVTD$_i$* as *SOURCE*, and *DVTD$_i$* as *DESTINATION*; let *EDESC$_i$* be the *DESCRIPTOR* returned from the application of those Syntax Rules.

e) *EDESC$_i$* is an edge table descriptor and describes an edge table of *SPG*.

7) Let *TPGD* be a tabular property graph descriptor that includes:

NOTE 158 — If <property graph content> is not specified, then *TPGD* is empty.

a) For *i*, 1 (one) ≤ *i* ≤ *n*, *VDESC$_i$*.

b) For *i*, 1 (one) ≤ *i* ≤ *m*, *EDESC$_i$*.

8) The Syntax Rules of Subclause 9.15, "Consistency check of a tabular property graph descriptor", are applied with *TPGD* as *TPGDESCRIPTOR*.

9) If the <property graph definition> is contained in a <schema definition>, then let *A* be the explicit or implicit <authorization identifier> of the <schema definition>; otherwise, let *A* be the <authorization identifier> that owns the schema identified by the explicit or implicit <schema name> of the <property graph name>.

Access Rules

1) If *PD* is contained in an <SQL-client module definition>, then the enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <property graph name>.

2) For each element table *T*:

a) Case:

i) If *PD* is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include SELECT on at least one column of *T*.

ii) Otherwise, the current privileges shall include SELECT on at least one column of *T*.

b) Let *C* be a column of *T* that is referenced in the <except column name list> or whose name is included in *TPGD*.

c) Case:

i) If *PD* is contained, without an intervening <SQL routine spec> that specifies SQL SECURITY INVOKER, in an <SQL schema statement>, then the applicable privileges of the <authorization identifier> that owns the containing schema shall include SELECT on *C*.

ii) Otherwise, the current privileges shall include SELECT on *C*.

General Rules

1) A <property graph definition> defines an SQL-property graph.

2) The General Rules of Subclause 9.16, "Deriving a pure property graph descriptor from a tabular property graph descriptor", are applied with *TPGD* as *TABULAR PROPERTY GRAPH DESCRIPTOR*; let *PPGD* be the *PURE PROPERTY GRAPH DESCRIPTOR* returned from the application of those General Rules.

3) An SQL-property graph descriptor *SPGD* is created that describes *SPG*. *SPGD* includes:

a) *SPN*.

b) *PPGD*.

c) *TPGD*.

4) A privilege descriptor is created that defines the privilege SELECT on *SPG* to *A*. That privilege is grantable if and only if the applicable privileges for *A* include all of the following:

a) Grantable SELECT privilege on every column that is referenced in any of the following:

i) The list of columns uniquely identifying a row in an edge table or a vertex table included in *TPGD*.

ii) The <value expression> associated with a property of a label of an edge table or a vertex table included in *TPGD*.

iii) The edge source key of an edge table included in *TPGD*.

iv) The source vertex key of an edge table included in *TPGD*.

v) The edge destination key of an edge table included in *TPGD*.

vi) The destination vertex key of an edge table included in *TPGD*.

b) Grantable EXECUTE privileges on every SQL-invoked routine that is a subject routine of <routine invocation>s contained in a <value expression> associated with a property of a label of an edge table or a vertex table included in *TPGD*.

c) Grantable SELECT privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in a <value expression> associated with a property of a label of an edge table or a vertex table included in *TPGD*, such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is the method identified by the <method name> of *MR*.

d) Grantable SELECT WITH HIERARCHY OPTION privilege on at least one supertable of the scoped table of every <reference resolution> that is contained in a <value expression> associated with a property of a label of an edge table or a vertex table included in *TPGD*.

The grantor of that privilege descriptor is set to the special grantor value "_SYSTEM".

Conformance Rules

1) Without Feature G920, “DDL-based SQL-property graphs”, conforming SQL language shall not contain a <property graph definition>.

2) Without Feature G921, “Empty SQL-property graph”, conforming SQL language shall not contain a <property graph definition> that does not specify <property graph content>.

3) Without Feature G922, “Views as element tables”, conforming SQL language shall not contain a <vertex table definition> that simply contains an <element table definition> that specifies an <element table name> that identifies a viewed table.

4) Without Feature G922, “Views as element tables”, conforming SQL language shall not contain an <edge table definition> that simply contains an <element table definition> that specifies an <element table name> that identifies a viewed table.

5) Without Feature G923, “In-line views as element tables”, conforming SQL language shall not contain a <vertex table definition> that simply contains an <element table definition> that immediately contains a <table subquery>.

6) Without Feature G923, “In-line views as element tables”, conforming SQL language shall not contain an <edge table definition> that simply contains an <element table definition> that immediately contains a <table subquery>.

7) Without Feature G924, “Explicit key clause for element tables”, conforming SQL language shall not contain an <element table key clause>.

8) Without Feature G924, “Explicit key clause for element tables”, conforming SQL language shall not contain a <source vertex table key clause> nor a <referenced source column list>.

9) Without Feature G924, “Explicit key clause for element tables”, conforming SQL language shall not contain a <destination vertex table key clause> nor a <referenced destination column list>.

10) Without Feature G925, “Explicit label and properties clause for element tables”, conforming SQL language shall not contain an <element table label and properties clause>.

11) Without Feature G926, “More than one label for vertex tables”, conforming SQL language shall not contain an <element table label and properties clause> simply contained in a <vertex table definition> that contains more than one <label and properties>.

12) Without Feature G927, “More than one label for edge tables”, conforming SQL language shall not contain an <element table label and properties clause> simply contained in an <edge table definition> that contains more than one <label and properties>.

13) Without Feature G928, “Value expressions as properties and renaming of properties”, conforming SQL language shall not contain a <derived property> that is not a <column reference>.

14) Without Feature G929, “Labels and properties: EXCEPT list”, conforming SQL language shall not contain an <except column name list>.

11.20 <alter property graph statement>

Function

Change the definition of an SQL-property graph.

Format

```
<alter property graph statement> ::=
  ALTER PROPERTY GRAPH <property graph name>
      <alter property graph action>

<alter property graph action> ::=
    <add element table definition>
  | <drop element table definition>
  | <alter element table definition>
```

Syntax Rules

1) Let *G* be the SQL-property graph identified by the <property graph name>.

2) The schema identified by the explicit or implicit <schema name> of the <property graph name> shall include the descriptor of *G*.

3) <alter property graph action> shall not generally contain a <graph reference> that identifies *G*.

Access Rules

1) The enabled authorization identifiers shall include the <authorization identifier> that owns the schema identified by the <schema name> of the SQL-property graph identified by <property graph name>.

General Rules

1) The SQL-property graph descriptor of *G* is modified as specified by the <alter property graph action>.

Conformance Rules

1) Without Feature G950, “Alter property graph statement: ADD/DROP element table”, conforming SQL language shall not contain an <alter property graph statement>.

11.21 <add element table definition>

Function

Add one or more vertex or edge tables to an SQL-property graph.

Format

```
<add element table definition> ::=
    ADD <vertex tables clause> [ ADD <edge tables clause> ]
  | ADD <edge tables clause>
```

Syntax Rules

1) Let *SPG* be the SQL-property graph identified by the <property graph name> immediately contained in the containing <alter property graph statement> *APGS*. Let *GD* be the descriptor of *SPG*.

2) Let *TPGD* be the tabular property graph descriptor included in *GD*.

3) If <vertex tables clause> *VTC* is specified, then let *n* be the number of <vertex table definition>s simply contained in *VTC*. For *i*, 1 (one) ≤ *i* ≤ *n*:

   a) Let VTD_i be the *i*-th <vertex table definition> simply contained in *VTC*. VTD_i shall not simply contain <source vertex table> or <destination vertex table>.

   b) The Syntax Rules of Subclause 9.13, "Creation of a vertex table descriptor", are applied with VTD_i as *BNFTERM*; let $VDESC_i$ be the *DESCRIPTOR* returned from the application of those Syntax Rules.

   c) $VDESC_i$ is a vertex table descriptor and describes a vertex table of *SPG*.

4) If <edge tables clause> *ETC* is specified, then let *m* be the number of <edge table definition>s simply contained in *ETC*. For *i*, 1 (one) ≤ *i* ≤ *m*:

   a) Let ETD_i be the *i*-th <edge table definition> simply contained in *ETC*. ETD_i shall simply contain a <source vertex table> SVT_i and a <destination vertex table> DVT_i.

   b) The <source vertex table alias> simply contained in SVT_i shall be equivalent to the element table alias included in a vertex table descriptor $SVTD_i$ either included in *TPGD* or instantiated in SR 3).

   c) The <destination vertex table alias> simply contained in DVT_i shall be equivalent to the element table alias included in a vertex table descriptor $DVTD_i$ either included in *TPGD* or instantiated in SR 3).

   d) The Syntax Rules of Subclause 9.14, "Creation of an edge table descriptor", are applied with ETD_i as *BNFTERM*, $SVTD_i$ as *SOURCE*, and $DVTD_i$ as *DESTINATION*; let $EDESC_i$ be the *DESCRIPTOR* returned from the application of those Syntax Rules.

   e) $EDESC_i$ is an edge table descriptor and describes an edge table of *SPG*.

5) Let *NTPGD* be a tabular property graph descriptor that is a copy of *TPGD* and additionally includes:

   a) For *i*, 1 (one) ≤ *i* ≤ *n*, $VDESC_i$.

   b) For *i*, 1 (one) ≤ *i* ≤ *m*, $EDESC_i$.

6) The Syntax Rules of Subclause 9.15, “Consistency check of a tabular property graph descriptor”, are applied with *NTPGD* as *TPGDESCRIPTOR*.

Access Rules

1) Let *A* be the <authorization identifier> that owns *SPG*.

2) For each element table *T* whose schema-qualified name is included in $VDESC_i$, 1 (one) ≤ *i* ≤ *n*, or in $EDESC_j$, 1 (one) ≤ *j* ≤ *m*, and that is a base table or a viewed table:

   a) The applicable privileges shall include:

      Case:

      i) If the privilege descriptor that defines the privilege SELECT on *SPG* to *A* with special grantor value "_SYSTEM" indicates that the privilege is grantable, then grantable SELECT on at least one column of *T*.

      ii) Otherwise, SELECT on at least one column of *T*.

   b) Let *C* be a column of *T* that is referenced in an <except column name list>, if any, or whose name is included in $VDESC_i$, for *i*, 1 (one) ≤ *i* ≤ *n*, or in $EDESC_j$, for *j*, 1 (one) ≤ *j* ≤ *m*.

   c) The applicable privileges shall include:

      Case:

      i) If the privilege descriptor that defines the privilege SELECT on *SPG* to *A* with special grantor value "_SYSTEM" indicates that the privilege is grantable, then grantable SELECT on *C*.

      ii) Otherwise, SELECT on *C*.

General Rules

1) The tabular property graph descriptor included in *GD* is replaced by *NTPGD*.

2) The General Rules of Subclause 9.16, “Deriving a pure property graph descriptor from a tabular property graph descriptor”, are applied with *NTPGD* as *TABULAR PROPERTY GRAPH DESCRIPTOR*; let *PPGD* be the *PURE PROPERTY GRAPH DESCRIPTOR* returned from the application of those General Rules.

3) The pure property graph descriptor included in *GD* is replaced by *PPGD*.

Conformance Rules

None.

11.22 <drop element table definition>

Function

Remove one or more vertex or edge tables from an SQL-property graph.

Format

```
<drop element table definition> ::=
  DROP <vertex or edge> TABLES <parenthesized element table alias list> <drop behavior>

<vertex or edge> ::=
    <vertex synonym>
  | <edge synonym>

<parenthesized element table alias list> ::=
  <left paren> <element table alias> [ { <comma> <element table alias> } ] <right paren>
```

Syntax Rules

1) Let *SPG* be the SQL-property graph identified by the <property graph name> immediately contained in the containing <alter property graph statement>. Let *GD* be the descriptor of *SPG*. Let *TPGD* be the tabular property graph descriptor included in *GD*.

2) Let *N* be the number of <element table alias>es simply contained in the <parenthesized element table alias list>. Let ET_i, 1 (one) ≤ *i* ≤ *N*, be an enumeration of those <element table alias>es. No <element table alias> shall be equivalent to any other <element table alias>.

3) For *i*, 1 (one) ≤ *i* ≤ *N*:

   Case:

   a) If <vertex or edge> immediately contains <vertex synonym>, then:

      i) ET_i shall be equivalent to the element table alias included in a vertex table descriptor included in *TPGD*.

      ii) If RESTRICT is specified, then ET_i shall not be equivalent to the name of the source vertex table or the name of the destination vertex table included in any edge table descriptor included in *TPGD*.

   b) Otherwise, ET_i shall be equivalent to the element table alias included in an edge table descriptor included in *TPGD*.

4) If RESTRICT is specified, then *SPG* shall not be referenced in any of the following:

   a) The original <query expression> of any view descriptor.

   b) The <search condition> of any constraint descriptor.

   c) The <search condition> of any assertion descriptor.

   d) The SQL routine body of any routine descriptor.

   e) The <parameter default> of any SQL parameter of any routine descriptor.

   f) The <triggered action> of any trigger descriptor.

g) Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

NOTE 159 — If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

None.

General Rules

1) Let *SOD* be the descriptor of a schema object dependent on *GD*.

NOTE 160 — A descriptor that "depends on" another descriptor is defined in Subclause 6.3.4, "Descriptors", in ISO/IEC 9075-1.

Case:

a) If *SOD* is a view descriptor, then let *SON* be the name of the view included in *SOD*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW SON CASCADE
```

b) If *SOD* is an assertion descriptor, then let *SON* be the name of the constraint included in *SOD*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION SON CASCADE
```

c) If *SOD* is a table constraint descriptor, then let *SON* be the name of the constraint included in *SOD*. Let *CTN* be the <table name> included in the table descriptor that includes *SOD*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE CTN DROP CONSTRAINT SON CASCADE
```

d) If *SOD* is a routine descriptor, then let *SON* be the specific name included in *SOD*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SON CASCADE
```

e) If *SOD* is a trigger descriptor, then let *SON* be the trigger name included in *SOD*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER SON CASCADE
```

f) If *SOD* is an SQL-property graph descriptor, then let *SON* be the name of the SQL-property graph included in *SOD*. The following <drop property graph statement> is effectively executed without further Access Rule checking:

```
DROP PROPERTY GRAPH SON CASCADE
```

2) For *i*, 1 (one) ≤ *i* ≤ *N*:

a) If <vertex or edge> immediately contains <vertex synonym>, then any edge table descriptor whose source vertex table name or destination vertex table name is equivalent to ET_i is removed from *TPGD*.

b) The element table descriptor whose element table alias is equivalent to ET_i is removed from *TPGD*.

3) The General Rules of Subclause 9.16, “Deriving a pure property graph descriptor from a tabular property graph descriptor”, are applied with *TPGD* as *TABULAR PROPERTY GRAPH DESCRIPTOR*; let *PPGD* be the *PURE PROPERTY GRAPH DESCRIPTOR* returned from the application of those General Rules.

4) The pure property graph descriptor included in *GD* is replaced with *PPGD*.

Conformance Rules

1) Without Feature G921, “Empty SQL-property graph”, in conforming SQL language, if <vertex or edge> immediately contains <vertex synonym>, then *TPGD* shall include *N*+1 (one) vertex table descriptors.

11.23 <alter element table definition>

Function

Change the options specified for a vertex or an edge table in an SQL-property graph.

Format

```
<alter element table definition> ::=
  ALTER <vertex or edge> TABLE <element table alias>
      <alter element table action>

<alter element table action> ::=
    <add element table label clause>
  | <drop element table label clause>
  | <alter element table label properties>
```

Syntax Rules

1) Let *SPG* be the SQL-property graph identified by the <property graph name> immediately contained in the containing <alter property graph statement>. Let *GD* be the descriptor of *SPG*.

2) If <vertex or edge> immediately contains <vertex synonym>, then *GD* shall include a vertex table descriptor that includes a vertex table alias that is equivalent to <element table alias>.

3) If <vertex or edge> immediately contains<edge synonym>, then *GD* shall include an edge table descriptor that includes an edge table alias that is equivalent to <element table alias>.

Access Rules

None.

General Rules

1) *GD* is modified as specified by the <alter element table action>.

Conformance Rules

None.

11.24 <add element table label clause>

Function

Add one or more labels to a vertex or an edge table in an SQL-property graph.

Format

```
<add element table label clause> ::=
  <add element table label>...

<add element table label> ::=
  ADD LABEL <label name> <element table properties clause>
```

Syntax Rules

1) Let *T* be the table identified by the <element table alias> *ETA* simply contained in the containing <alter element table definition>. Let *TD* be the element table descriptor that is identified by *ETA*.

2) Let *TPGD* be the tabular property graph descriptor that includes *TD*.

3) Let *N* be the number of <add element table label>s simply contained in <add element table label clause>.

4) For each *i*, 1 (one) ≤ *i* ≤ *N*:

   a) Let LN_i be the <label name> simply contained in the *i*-th <add element table label>.

   b) LN_i shall not be equivalent to a label name included in *TD* or the <label name> included in the *k*-th <add element table label>, 1 (one) ≤ *k* ≤ *N*, and *i* ≠ *k*.

   c) Case:

      i) If NO PROPERTIES is specified, then let P_i be 0 (zero).

      ii) If <element table parenthesized derived property list> is specified, then let P_i be the number of simply contained <derived property>s. For *j*, 1 (one) ≤ *j* ≤ P_i:

         1) Every <column reference> simply contained in the *j*-th <derived property> shall reference a column of *T*.

         2) If the *j*-th <derived property> immediately contains a <property name>, then let $PN_{i,j}$ be that <property name>; otherwise, the <value expression> immediately contained in the *j*-th <derived property> shall be a <column reference> and let $PN_{i,j}$ be the name of the column referenced by that <column reference>.

         3) Let $PT_{i,j}$ be the declared type of the <value expression> immediately contained in the *j*-th <derived property>.

         4) Let $PV_{i,j}$ be the <value expression> immediately contained in the *j*-th <derived property>.

         5) $PV_{i,j}$ shall not contain a potential source of non-determinism.

         6) $PV_{i,j}$ shall not generally contain a <routine invocation> whose subject routine possibly reads SQL-data.

7) $PV_{i,j}$ shall not contain a <query expression>.

iii) If EXCEPT is not specified, then let P_i be the number of columns of T. For j, 1 (one) $\leq j \leq P_i$:

1) Let $PN_{i,j}$ be the name of the j-th column of T.

2) Let $PT_{i,j}$ be the declared type of the j-th column of T.

3) Let $PV_{i,j}$ be the name of the j-th column of T.

iv) Otherwise, let TC be the number of columns of T, let EC_i be the number of <column name>s simply contained in the <except column name list> $ECNL_i$, and let P_i be $TC_i - EC_i$.

1) No <column name> simply contained in $ECNL_i$ shall be equivalent to another <column name> simply contained in $ECNL_i$.

2) Each <column name> simply contained in $ECNL_i$ shall be equivalent to the name of a column of T.

3) Let RC be the set of columns of T whose column name is not equivalent to a <column name> simply contained in $ECNL_i$ (there are P_i such columns).

4) For j, 1 (one) $\leq j \leq P_i$:

A) Let $PN_{i,j}$ be the name of the j-th column in RC.

B) Let $PT_{i,j}$ be the declared type of the j-th column in RC.

C) Let $PV_{i,j}$ be the name of the j-th column in RC.

d) For each pair of implicit or explicit properties (P_s, P_t), 1 (one) $\leq s \leq P_i$, 1 (one) $\leq t \leq P_i$, $s \neq t$, the name of P_s shall not be equivalent to the name of P_t.

5) For each pair of implicit or explicit properties ($P_{k,s}$, $P_{l,t}$), 1 (one) $\leq k \leq N$, 1 (one) $\leq l \leq N$, $k \neq l$, 1 (one) $\leq s \leq P_k$, 1 (one) $\leq t \leq P_l$:

a) If $PN_{k,s}$ is equivalent to $PN_{l,t}$, then $PT_{k,s}$ shall be the same as $PT_{l,t}$, and $PV_{k,s}$ shall be the same as $PV_{l,t}$.

b) If $PN_{k,s}$ is equivalent to the name of a property EP included in TD, then $PT_{k,s}$ shall be the same as the declared type of EP, and $PV_{k,s}$ shall be the same as the value expression associated with EP.

6) Let NTD be an element table descriptor that is a copy of TD with the addition of N new labels to the set of labels as follows. For each label, 1 (one) $\leq j \leq N$:

a) The name of the label: TL_j.

b) The set of properties associated with the label. For each property, 1 (one) $\leq k \leq P_j$:

i) The name of the property: $PN_{j,k}$.

ii) The declared type of the property: $PT_{j,k}$.

iii) The value expression associated with the property: $PV_{j,k}$.

c) Let *NTPGD* be a tabular property graph descriptor that is a copy of *TPGD* in which *TD* is replaced by *NTD*. The Syntax Rules of Subclause 9.15, "Consistency check of a tabular property graph descriptor", are applied with NTPGD as *TPGDESCRIPTOR*.

Access Rules

1) Let *SPG* be the SQL-property graph that is described by the SQL-property graph descriptor that includes *TPGD*. Let *A* be the <authorization identifier> that owns *SPG*.

2) If *T* is a base table or a viewed table, then:

   a) Let *C* be a column of *T* that is referenced in an <except column name list>, if any, or whose name is included in $PV_{j,k}$, for *j*, 1 (one) ≤ *j* ≤ *N*, and *k*, 1 (one) ≤ *k* ≤ P_j.

   b) The applicable privileges shall include:

      Case:

      i) If the privilege descriptor that defines the privilege SELECT on *SPG* to *A* with special grantor value "_SYSTEM" indicates that the privilege is grantable, then grantable SELECT on *C*.

      ii) Otherwise, SELECT on *C*.

General Rules

1) Let *SPGD* be the SQL-property graph descriptor that includes *TPGD*. *TPGD* included in *SPGD* is replaced by *NTPGD*.

2) The General Rules of Subclause 9.16, "Deriving a pure property graph descriptor from a tabular property graph descriptor", are applied with *NTPGD* as *TABULAR PROPERTY GRAPH DESCRIPTOR*; let *PPGD* be the *PURE PROPERTY GRAPH DESCRIPTOR* returned from the application of those General Rules.

3) The pure property graph descriptor included in *SPGD* is replaced by *PPGD*.

Conformance Rules

1) Without Feature G926, "More than one label for vertex tables", in conforming SQL language, *TD* shall not be a vertex table descriptor.

2) Without Feature G927, "More than one label for edge tables", in conforming SQL language, *TD* shall not be an edge table descriptor.

3) Without Feature G960, "Alter element table definition: ADD/DROP LABEL", conforming SQL language shall not contain an <add element table label clause>.

11.25 <drop element table label clause>

Function

Drop a label from a vertex or an edge table in an SQL-property graph.

Format

```
<drop element table label clause> ::=
  DROP LABEL <label name> <drop behavior>
```

Syntax Rules

1) Let *T* be the table identified by the <element table alias> simply contained in the containing <alter element table definition>.

2) Let *SPG* be the SQL-property graph identified by the <property graph name> immediately contained in the containing <alter property graph statement>.

3) Let *GD* be the descriptor of *SPG*.

4) Let *TPGD* be the tabular property graph descriptor included in *GD*.

5) Let *TD* be the element table descriptor included in *TPGD* that describes *T*.

6) *TD* shall include two or more labels, one of which has an <identifier> that is equivalent to the <label name> simply contained in the <drop element table label clause>.

7) If RESTRICT is specified, then *SPG* shall not be referenced in any of the following:

   a) The original <query expression> of any view descriptor.

   b) The <search condition> of any constraint descriptor.

   c) The <search condition> of any assertion descriptor.

   d) The SQL routine body of any routine descriptor.

   e) The <parameter default> of any SQL parameter of any routine descriptor.

   f) The <triggered action> of any trigger descriptor.

   g) Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

   NOTE 161 — If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

Access Rules

None.

General Rules

1) Let *SOD* be the descriptor of a schema object dependent on *GD*.

Case:

a) If *SOD* is a view descriptor, then let *SON* be the name of the view included in *SOD*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW SON CASCADE
```

b) If *SOD* is an assertion descriptor, then let *SON* be the name of the constraint included in *SOD*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION SON CASCADE
```

c) If *SOD* is a table constraint descriptor, then let *SON* be the name of the constraint included in *SOD*. Let *CTN* be the <table name> included in the table descriptor that includes *SOD*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE CTN DROP CONSTRAINT SON CASCADE
```

d) If *SOD* is a routine descriptor, then let *SON* be the specific name included in *SOD*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SON CASCADE
```

e) If *SOD* is a trigger descriptor, then let *SON* be the trigger name included in *SOD*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER SON CASCADE
```

f) If *SOD* is an SQL-property graph descriptor, then let *SON* be the name of the SQL-property graph included in *SOD*. The following <drop property graph statement> is effectively executed without further Access Rule checking:

```
DROP PROPERTY GRAPH SON CASCADE
```

2) The label identified by <label name> is removed from *TD*.

3) The General Rules of Subclause 9.16, "Deriving a pure property graph descriptor from a tabular property graph descriptor", are applied with *TPGD* as *TABULAR PROPERTY GRAPH DESCRIPTOR*; let *PPGD* be the *PURE PROPERTY GRAPH DESCRIPTOR* returned from the application of those General Rules.

4) The pure property graph descriptor included in *GD* is replaced with *PPGD*.

Conformance Rules

1) Without Feature G926, "More than one label for vertex tables", in conforming SQL language, *TD* shall not be a vertex table descriptor.

2) Without Feature G927, "More than one label for edge tables", in conforming SQL language, *TD* shall not be an edge table descriptor.

3) Without Feature G960, "Alter element table definition: ADD/DROP LABEL", conforming SQL language shall not contain a <drop element table label clause>.

11.26 <alter element table label properties>

Function

Add or drop properties from a label of a vertex or an edge table in an SQL-property graph.

Format

```
<alter element table label properties> ::=
  ALTER LABEL <label name> <alter label action>

<alter label action> ::=
    <add property definition>
  | <drop property definition>

<add property definition> ::=
  ADD PROPERTIES <element table parenthesized derived property list>

<drop property definition> ::=
  DROP PROPERTIES <parenthesized property name list> <drop behavior>

<parenthesized property name list> ::=
  <left paren> <property name list> <right paren>

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

Syntax Rules

1) Let *T* be the table identified by the <element table alias> simply contained in the containing <alter element table definition>.

2) Let *SPG* be the SQL-property graph identified by the <property graph name> immediately contained in the containing <alter property graph statement>.

3) Let *GD* be the descriptor of *SPG*.

4) Let *TPGD* be the tabular property graph descriptor included in *GD*.

5) Let *TD* be the element table descriptor included in *TPGD* that describes *T*.

6) *TD* shall include a label *L* whose <identifier> is equivalent to the <label name> simply contained in the <alter element table label properties>.

7) If <alter label action> immediately contains <add property definition>, then:

   a) Let *NP* be the number of <derived property>s simply contained in the <element table parenthesized derived property list>.

   b) For *i*, 1 (one) ≤ *i* ≤ *NP*:

      i) Every <column reference> simply contained in the *i*-th <derived property> shall reference a column of *T*.

      ii) Case:

         1) If the *i*-th <derived property> simply contains a <property name>, then let PN_i be that <property name>.

2) Otherwise, the <value expression> simply contained in the *i*-th <derived property> shall be a <column reference>. Let PN_i be the name of the column referenced by that <column reference>.

iii) *L* shall not include a property whose name is equivalent to PN_i.

iv) Let PT_i be the declared type of the <value expression> simply contained in the *i*-th <derived property>.

v) Let PV_i be the <value expression> simply contained in the *i*-th <derived property>.

vi) PV_i shall not contain a potential source of non-determinism.

vii) PV_i shall not generally contain a <routine invocation> whose subject routine possibly reads SQL-data.

viii) PV_i shall not contain a <query expression>.

ix) If *TD* includes a label *K* other than *L*, and *K* has a property *KP* whose name is equivalent to PN_i, then the <value expression> associated with *KP* shall have the same left normal form derivation as PV_i.

NOTE 162 — "Left normal form derivation" is defined in Subclause 6.2, "Notation provided in the ISO/IEC 9075 series", in ISO/IEC 9075-1.

NOTE 163 — The preceding Syntax Rule ensures that two properties with the same name within the same element table but in different labels are the same; i.e., have the same definition and, by implication, the same declared type.

c) Let *LL* be a copy of *L* with *NP* new properties. For *i*, 1 (one) ≤ *i* ≤ *NP*, the *i*-th new property is:

i) The name of the property is PN_i.

ii) The declared type of the property is PT_i.

iii) The value expression associated with the property is PV_i.

8) If <alter label action> immediately contains <drop property definition>, then:

a) If RESTRICT is specified, then *SPG* shall not be referenced in any of the following:

i) The original <query expression> of any view descriptor.

ii) The <search condition> of any constraint descriptor.

iii) The <search condition> of any assertion descriptor.

iv) The SQL routine body of any routine descriptor.

v) The <parameter default> of any SQL parameter of any routine descriptor.

vi) The <triggered action> of any trigger descriptor.

vii) Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

NOTE 164 — If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

b) Let *DP* be the number of <property name>s simply contained in the <parenthesized property name list>.

c) For *i*, 1 (one) ≤ *i* ≤ *NP*, let DP_i be an enumeration of those property names. *L* shall have a property whose name is equivalent to DP_i.

d) Let *LL* be a copy of *L* where for *i*, 1 (one) ≤ *i* ≤ *NP*, the property whose name is equivalent to DP_i is removed from *LL*.

9) Let *TDN* be copy of *TD*, in which *L* is replaced by *LL*.

10) Let *NTPGD* be copy of *TPGD*, in which *TD* is replaced by *TDN*.

11) The Syntax Rules of Subclause 9.15, "Consistency check of a tabular property graph descriptor", are applied with *NTPGD* as *TPGDESCRIPTOR*.

Access Rules

1) If <add property definition> is specified and *T* is a base table or a viewed table, then let *C* be a column of *T* that is referenced in the <element table parenthesized derived property list>. Let *A* be the <authorization identifier> that owns *SPG*. The applicable privileges shall include:

Case:

a) If the privilege descriptor that defines the privilege SELECT on *SPG* to *A* with special grantor value "_SYSTEM" indicates that the privilege is grantable, then grantable SELECT on *C*.

b) Otherwise, SELECT on *C*.

General Rules

1) If <alter label action> immediately contains <drop property definition>, then let *SOD* be the descriptor of a schema object dependent on *GD*.

Case:

a) If *SOD* is a view descriptor, then let *SON* be the name of the view included in *SOD*. The following <drop view statement> is effectively executed without further Access Rule checking:

```
DROP VIEW SON CASCADE
```

b) If *SOD* is an assertion descriptor, then let *SON* be the name of the constraint included in *SOD*. The following <drop assertion statement> is effectively executed without further Access Rule checking:

```
DROP ASSERTION SON CASCADE
```

c) If *SOD* is a table constraint descriptor, then let *SON* be the name of the constraint included in *SOD*. Let *CTN* be the <table name> included in the table descriptor that includes *SOD*. The following <alter table statement> is effectively executed without further Access Rule checking:

```
ALTER TABLE CTN DROP CONSTRAINT SON CASCADE
```

d) If *SOD* is a routine descriptor, then let *SON* be the specific name included in *SOD*. The following <drop routine statement> is effectively executed without further Access Rule checking:

```
DROP SPECIFIC ROUTINE SON CASCADE
```

e) If *SOD* is a trigger descriptor, then let *SON* be the trigger name included in *SOD*. The following <drop trigger statement> is effectively executed without further Access Rule checking:

```
DROP TRIGGER SON CASCADE
```

f) If *SOD* is an SQL-property graph descriptor, then let *SON* be the name of the SQL-property graph included in *SOD*. The following <drop property graph statement> is effectively executed without further Access Rule checking:

```
DROP PROPERTY GRAPH SON CASCADE
```

NOTE 165 — A descriptor that "depends on" another descriptor is defined in Subclause 6.3.4, "Descriptors", in ISO/IEC 9075-1.

2) The General Rules of Subclause 9.16, "Deriving a pure property graph descriptor from a tabular property graph descriptor", are applied with *NTPGD* as *TABULAR PROPERTY GRAPH DESCRIPTOR*; let *PPGD* be the *PURE PROPERTY GRAPH DESCRIPTOR* returned from the application of those General Rules.

3) The pure property graph descriptor included in *GD* is replaced with *PPGD*.

4) The tabular property graph descriptor included in *GD* is replaced with *NTPGD*.

Conformance Rules

1) Without Feature G970, "Alter element table definition: ALTER LABEL" conforming SQL language shall not contain an <alter element table label properties>.

11.27 <drop property graph statement>

Function

Destroy an SQL-property graph.

Format

```
<drop property graph statement> ::=
  DROP PROPERTY GRAPH <property graph name> <drop behavior>
```

Syntax Rules

1) Let *PG* be the SQL-property graph identified by the <property graph name> *PGN*. The schema identified by the explicit or implicit <schema name> of *PGN* shall include the descriptor of *PG*.

2) If RESTRICT is specified, then *PG* shall not be referenced in any of the following:

   a) The original <query expression> of any view descriptor.

   b) The <search condition> of any constraint descriptor.

   c) The <search condition> of any assertion descriptor.

   d) The SQL routine body of any routine descriptor.

   e) The <parameter default> of any SQL parameter of any routine descriptor.

   f) The <triggered action> of any trigger descriptor.

   g) Any <value expression> associated with the <query expression> or a property of any label, either of which are included in any element table descriptor included in the tabular property graph descriptor included in any SQL-property graph descriptor.

   NOTE 166 — If CASCADE is specified, then any such dependent object will be dropped by the execution of the <revoke statement> specified in the General Rules of this Subclause.

3) Let *A* be the <authorization identifier> that owns the schema identified by the <schema name> of the SQL-property graph identified by *PGN*.

Access Rules

1) The enabled authorization identifiers shall include *A*.

General Rules

1) The following <revoke statement> is effectively executed with a current authorization identifier of "_SYSTEM" and without further Access Rule checking:

   ```
   REVOKE ALL PRIVILEGES ON PROPERTY GRAPH PGN FROM A CASCADE
   ```

2) The descriptor of *PG* is destroyed.

Conformance Rules

1) Without Feature G920, “DDL-based SQL-property graphs” conforming SQL language shall not contain a <drop property graph statement>.

2) Without Feature G980, “DROP PROPERTY GRAPH: CASCADE drop behavior” conforming SQL language shall not contain a <drop property graph statement> that contains <drop behavior> that contains CASCADE.

12 Access control

This Clause modifies Clause 12, "Access control", in ISO/IEC 9075-2.

12.1 <grant statement>

This Subclause modifies Subclause 12.1, "<grant statement>", in ISO/IEC 9075-2.

Function

Define privileges and role authorizations.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 7): For every involved grantee *G*, if following the successful execution of the <grant statement>, the applicable privileges for *G* include grantable SELECT privilege on an SQL-property graph identified by its <property graph name> *PG*, then the following <grant statement> is effectively executed as though the current user identifier were "_SYSTEM" and without further Access Rule checking:

```
GRANT SELECT
   ON PG
   TO G
   WITH GRANT OPTION
```

Conformance Rules

No additional Conformance Rules.

12.2 <privileges>

This Subclause modifies Subclause 12.3, “<privileges>”, in ISO/IEC 9075-2.

Function

Specify privileges.

Format

```
<object name> ::=
    !! All alternatives from ISO/IEC 9075-2
  | PROPERTY GRAPH <property graph name>
```

Syntax Rules

1) Insert after SR 4)b): If *ON* specifies a <property graph name>, then *AC* shall specify SELECT.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

1) Insert after the last CR: Without Feature G920, “DDL-based SQL-property graphs”, conforming SQL language shall not contain an <object name> that contains PROPERTY GRAPH.

12.3 <revoke statement>

This Subclause modifies Subclause 12.7, "<revoke statement>", in ISO/IEC 9075-2.

Function

Destroy privileges and role authorizations.

Format

No additional Format items.

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

1) Insert after GR 2)b)i)3)B): *P* and *D* are both SQL-property graph privilege descriptors. The action and the identified SQL-property graph of *P* are the same as the action and the identified SQL-property graph of *D*, respectively.

2) Insert after GR 12)a)i)12): SELECT privilege on every SQL-property graph identified by a <graph reference> contained in *QE*.

3) Insert after GR 12)b): Let *PG* be any SQL-property graph descriptor included in *S1*. Let *TPGD* be the tabular property graph descriptor included in *PG*. *PG* is said to be *abandoned* if, for any edge table descriptor *ET* or vertex table descriptor *ET* included in *TPGD*, the revoke destruction action would result in *A1* no longer having in its applicable privileges any of the following:

   a) Case:

      i) If *ET* contains a <query expression> *QE*, then:

         1) SELECT privilege on at least one column of each table identified by a <table reference> contained in *QE*.

         2) SELECT privilege on any column identified by a <column reference> contained in *QE*.

         3) USAGE privilege on every domain, every collation, every character set, and every transliteration whose names are contained in *QE*.

         4) USAGE privilege on any user-defined type *UDT* such that some data type included in *QE* is usage-dependent on *UDT*.

         5) The required execute privileges of *QE*.

         6) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in *QE* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is subject routine of *MR*.

7) SELECT privilege on any column identified by a <column reference> contained in the <scalar subquery> that is equivalent to some <dereference operation> contained in *QE*.

8) SELECT WITH HIERARCHY OPTION privilege on at least one supertable of the scoped table of any <reference resolution> that is contained in *QE*.

9) SELECT privilege on the scoped table of any <reference resolution> that is contained in *QE*.

10) SELECT WITH HIERARCHY OPTION privilege on at least one supertable of every typed table identified by a <table reference> that simply contains an <only spec> and that is contained in *QE*.

11) SELECT privilege on every SQL-property graph identified by a <graph reference> contained in *QE*.

ii) Otherwise, SELECT privilege on at least one column of the table identified as the edge table or vertex table of *ET*.

b) SELECT privilege on any column that uniquely identifies a row in *ET*.

c) If *ET* is an edge table descriptor, then SELECT privilege on any column contained in the edge source key, source vertex key, edge destination key, or destination vertex key of *ET*.

d) For any <value expression> *VE* of any property of any label of *ET*:

i) SELECT privilege on at least one column of each table identified by a <table reference> contained in *VE*.

ii) SELECT privilege on any column identified by a <column reference> contained in *VE*.

iii) USAGE privilege on every domain, every collation, every character set, and every transliteration whose names are contained in *VE*.

iv) USAGE privilege on any user-defined type *UDT* such that some data type included in *VE* is usage-dependent on *UDT*.

v) The required execute privileges of *VE*.

vi) The table/method privilege on every table *T1* and every method *M* such that there is a <method reference> *MR* contained in *VE* such that *T1* is in the scope of the <value expression primary> of *MR* and *M* is subject routine of *MR*.

vii) SELECT privilege on any column identified by a <column reference> contained in the <scalar subquery> that is equivalent to some <dereference operation> contained in *VE*.

viii) SELECT WITH HIERARCHY OPTION privilege on at least one supertable of the scoped table of any <reference resolution> that is contained in *VE*.

ix) SELECT privilege on the scoped table of any <reference resolution> that is contained in *VE*.

x) SELECT WITH HIERARCHY OPTION privilege on at least one supertable of every typed table identified by a <table reference> that simply contains an <only spec> and that is contained in *VE*.

xi) SELECT privilege on every SQL-property graph identified by a <graph reference> contained in *VE*.

4) Insert after GR 12)c)i)10): SELECT privilege on every SQL-property graph identified by a <graph reference> contained in the applicable <search condition> of *TC*.

5) Insert after GR 12)d)i)10): SELECT privilege on every SQL-property graph identified by a <graph reference> contained in the applicable <search condition> of *AX*.

6) Insert after GR 12)e)i)23): SELECT privilege on every SQL-property graph identified by a <graph reference> contained in any <search condition> of *TR*.

7) Insert after GR 12)f)i)10): SELECT privilege on every SQL-property graph identified by a <graph reference> contained in the <search condition> of *DC*.

8) Insert after GR 12)o)i)19): SELECT privilege on every SQL-property graph identified by a <graph reference> contained in the <SQL routine body> of *RD*.

9) Insert after GR 12)q): If RESTRICT is specified, and there exists an abandoned SQL-property graph descriptor, then an exception condition is raised: *dependent privilege descriptors still exist (2B000)*.

10) Insert after GR 12)ah): For every abandoned SQL-property graph descriptor *PG*, let *S1.PN* be the <property graph name> of *PG*. The following <drop property graph statement> is effectively executed without further Access Rule checking:

```
DROP PROPERTY GRAPH S1.PN CASCADE
```

Conformance Rules

No additional Conformance Rules.

13 SQL-client modules

This Clause modifies Clause 13, "SQL-client modules", in ISO/IEC 9075-2.

13.1 <externally-invoked procedure>

This Subclause modifies Subclause 13.3, "<externally-invoked procedure>", in ISO/IEC 9075-2.

Function

Define an externally-invoked procedure.

Format

No additional Format items.

Syntax Rules

1) Insert into SR 10)e) after

```
    package SQLSTATE_CODES is
```

the code:

```
    DATA_EXCEPTION_NO_SUBCLASS:
      constant SQLSTATE_TYPE := "22000";
    DATA_EXCEPTION_INVALID_NUMBER_OF_PATHS_OR_GROUPS:
      constant SQLSTATE_TYPE := "22G0F";
    DATA_EXCEPTION_MULTI_SOURCED_OR_MULTI_DESTINED_EDGE:
      constant SQLSTATE_TYPE := "22G0K";
    DATA_EXCEPTION_INCOMPLETE_EDGE:
      constant SQLSTATE_TYPE := "22G0L";
```

The text of the Ada library unit package Interfaces.SQL is also available from the ISO website as a "digital artifact". See `https://standards.iso.org/iso-iec/9075/-16/ed-1/en/` to download digital artifacts for this document. To download the library unit package, select the file named `ISO_IEC_9075-16(E)_PGQ-Interfaces.SQL.ada`.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

13.2 <SQL procedure statement>

This Subclause modifies Subclause 13.4, "<SQL procedure statement>", in ISO/IEC 9075-2.

Function

Define all of the SQL-statements that are <SQL procedure statement>s.

Format

```
<SQL schema definition statement> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <property graph definition>

<SQL schema manipulation statement> ::=
    !! All alternatives from ISO/IEC 9075-2
  | <alter property graph statement>
  | <drop property graph statement>
```

Syntax Rules

No additional Syntax Rules.

Access Rules

No additional Access Rules.

General Rules

No additional General Rules.

Conformance Rules

No additional Conformance Rules.

14 Diagnostics management

This Clause modifies Clause 23, "Diagnostics management", in ISO/IEC 9075-2.

14.1 <get diagnostics statement>

This Subclause modifies Subclause 23.1, "<get diagnostics statement>", in ISO/IEC 9075-2.

Function

Get exception or completion condition information from a diagnostics area.

Format

```
<condition information item name> ::=
    !! All alternatives from ISO/IEC 9075-2
  | PROPERTY_GRAPH_CATALOG
  | PROPERTY_GRAPH_NAME
  | PROPERTY_GRAPH_SCHEMA
```

Syntax Rules

1) Insert into Table 38, "Data types of <condition information item name>s" the rows of Table 2, "Data types of <condition information item name>s".

Table 2 — Data types of <condition information item name>s

<identifier>	Declared Type
PROPERTY_GRAPH_CATALOG	variable-length character string with implementation-defined (IL005) maximum length
PROPERTY_GRAPH_NAME	variable-length character string with implementation-defined (IL005) maximum length
PROPERTY_GRAPH_SCHEMA	variable-length character string with implementation-defined (IL005) maximum length

Access Rules

No additional Access Rules.

General Rules

1) Insert into Table 39, "SQL-statement codes" the rows of Table 3, "SQL-statement codes".

Table 3 — SQL-statement codes

SQL-statement	Identifier	Code
<alter property graph statement>	ALTER PROPERTY GRAPH	140
<drop property graph statement>	DROP PROPERTY GRAPH	141
<property graph definition>	CREATE PROPERTY GRAPH	142

2) Insert before GR 4)h)i)2): If the syntax error or access rule violation was caused by reference to a specific SQL-property graph, then the values of PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, and PROPERTY_GRAPH_NAME are the <catalog name>, the <unqualified schema name> of the <schema name> of the schema that contains the SQL-property graph that caused the syntax error or access rule violation, and the <qualified identifier> of that SQL-property graph, respectively.

3) Augment GR 4)h)i)2) by adding "PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, and PROPERTY_GRAPH_NAME" to the list that contain the zero-length character string.

4) Insert before GR 4)k)iv): If the values of PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, and PROPERTY_GRAPH_NAME identify an SQL-property graph for which no privileges are granted to the enabled authorization identifiers, then the values of PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, and PROPERTY_GRAPH_NAME are replaced by the zero-length character string.

Conformance Rules

1) Insert after the last CR: Without Feature G860, "GET DIAGNOSTICS enhancements for SQL-property graphs", conforming SQL language shall not contain a <condition information item name> that contains PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_NAME, or PROPERTY_GRAPH_SCHEMA.

15 Information Schema

This Clause modifies Clause 6, "Information Schema", in ISO/IEC 9075-11.

15.1 Information Schema digital artifact

This Subclause modifies Subclause 6.1, "Information Schema digital artifact", in ISO/IEC 9075-11.

Insert after the 1st paragraph: These schema definition and manipulation statements are also available from the ISO website as a "digital artifact". See `https://standards.iso.org/iso-iec/9075/-16/ed-1/en/` to download digital artifacts for this document. To download the schema definition and manipulation statements, select the file named `ISO_IEC_9075-16(E)_PGQ-schema-definition.sql`.

15.2 PG_DEFINED_LABEL_SETS view

Function

Identify the defined label sets in an SQL-property graph defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PG_DEFINED_LABEL_SETS AS
    SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
           DEFINED_LABEL_SET_IDENTIFIER
    FROM DEFINITION_SCHEMA.PG_DEFINED_LABEL_SETS
    WHERE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME ) IN
            ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                     PGP.PROPERTY_GRAPH_NAME
              FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
              WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                     OR
                      PGP.GRANTEE IN ( SELECT ROLE_NAME FROM ENABLED_ROLES ) ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_DEFINED_LABEL_SETS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, "SQL/PGQ Information Schema views", conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_DEFINED_LABEL_SETS.

15.3 PG_DEFINED_LABEL_SET_LABELS view

Function

Identify the labels of each defined label set in an SQL-property graph defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PG_DEFINED_LABEL_SET_LABELS AS
    SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
           DEFINED_LABEL_SET_IDENTIFIER, LABEL_NAME
    FROM DEFINITION_SCHEMA.PG_DEFINED_LABEL_SET_LABELS
    WHERE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME ) IN
            ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                     PGP.PROPERTY_GRAPH_NAME
              FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
              WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                     OR
                      PGP.GRANTEE IN ( SELECT ROLE_NAME
                                       FROM ENABLED_ROLES ) ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_DEFINED_LABEL_SET_LABELS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_DEFINED_LABEL_SET_LABELS.

15.4 PG_EDGE_DEFINED_LABEL_SETS view

Function

Identify the edge defined label sets in an SQL-property graph defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PG_EDGE_DEFINED_LABEL_SETS AS
    SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
           EDGE_DEFINED_LABEL_SET_IDENTIFIER, IS_DIRECTED, IS_UNDIRECTED
    FROM DEFINITION_SCHEMA.PG_EDGE_DEFINED_LABEL_SETS
    WHERE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME ) IN
            ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                     PGP.PROPERTY_GRAPH_NAME
              FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
              WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                     OR
                      PGP.GRANTEE IN ( SELECT ROLE_NAME
                                       FROM ENABLED_ROLES ) ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_EDGE_DEFINED_LABEL_SETS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_EDGE_DEFINED_LABEL_SETS.

15.5 PG_EDGE_TABLE_COMPONENTS view

Function

Identify the columns that are part of an edge destination key/destination vertex key or edge source key/source vertex key of an edge table in an SQL-property graph that is owned by a given user or role and is defined in this catalog.

Definition

```
CREATE VIEW PG_EDGE_TABLE_COMPONENTS AS
    SELECT ETC.PROPERTY_GRAPH_CATALOG, ETC.PROPERTY_GRAPH_SCHEMA,
           ETC.PROPERTY_GRAPH_NAME, ETC.EDGE_TABLE_ALIAS, ETC.VERTEX_TABLE_ALIAS,
           ETC.EDGE_END, ETC.EDGE_TABLE_COLUMN_NAME, ETC.VERTEX_TABLE_COLUMN_NAME,
           ETC.ORDINAL_POSITION
    FROM DEFINITION_SCHEMA.PG_EDGE_TABLE_COMPONENTS AS ETC
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
        ON ( ( ETC.PROPERTY_GRAPH_CATALOG, ETC.PROPERTY_GRAPH_SCHEMA )
           = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
           OR
            S.SCHEMA_OWNER IN ( SELECT ER.ROLE_NAME
                                FROM ENABLED_ROLES AS ER ) )
       AND
          ETC.PROPERTY_GRAPH_CATALOG
        = ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN );

GRANT SELECT ON TABLE PG_EDGE_TABLE_COMPONENTS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_EDGE_TABLE_COMPONENTS.

15.6 PG_EDGE_TRIPLETS view

Function

Identify the edge triplets in an SQL-property graph defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PG_EDGE_TRIPLETS AS
    SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
           EDGE_TRIPLET_DEFINED_LABEL_SET_IDENTIFIER,
           SOURCE_VERTEX_DEFINED_LABEL_SET_IDENTIFIER,
           DESTINATION_VERTEX_DEFINED_LABEL_SET_IDENTIFIER, IS_DIRECTED
    FROM DEFINITION_SCHEMA.PG_EDGE_TRIPLETS
    WHERE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME ) IN
            ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                     PGP.PROPERTY_GRAPH_NAME
              FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
              WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                     OR
                      PGP.GRANTEE IN ( SELECT ROLE_NAME
                                       FROM ENABLED_ROLES ) ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_EDGE_TRIPLETS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_EDGE_TRIPLETS.

15.7 PG_ELEMENT_TABLE_KEY_COLUMNS view

Function

Identify the columns that identify a row in an element table in an SQL-property graph that is owned by a given user or role and is defined in this catalog.

Definition

```
CREATE VIEW PG_ELEMENT_TABLE_KEY_COLUMNS AS
    SELECT ETKC.PROPERTY_GRAPH_CATALOG, ETKC.PROPERTY_GRAPH_SCHEMA,
           ETKC.PROPERTY_GRAPH_NAME, ETKC.ELEMENT_TABLE_ALIAS, ETKC.COLUMN_NAME,
           ETKC.ORDINAL_POSITION
    FROM DEFINITION_SCHEMA.PG_ELEMENT_TABLE_KEY_COLUMNS AS ETKC
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
        ON ( ( ETKC.PROPERTY_GRAPH_CATALOG, ETKC.PROPERTY_GRAPH_SCHEMA )
           = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
           OR
            S.SCHEMA_OWNER IN ( SELECT ER.ROLE_NAME
                                FROM ENABLED_ROLES AS ER ) )
       AND
          ETKC.PROPERTY_GRAPH_CATALOG
        = ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN );

GRANT SELECT ON TABLE PG_ELEMENT_TABLE_KEY_COLUMNS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_ELEMENT_TABLE_KEY_COLUMNS.

15.8 PG_ELEMENT_TABLE_LABELS view

Function

Identify the labels associated with an element table in an SQL-property graph that is owned by a given user or role and is defined in this catalog.

Definition

```
CREATE VIEW PG_ELEMENT_TABLE_LABELS AS
    SELECT ETL.PROPERTY_GRAPH_CATALOG, ETL.PROPERTY_GRAPH_SCHEMA,
           ETL.PROPERTY_GRAPH_NAME, ETL.ELEMENT_TABLE_ALIAS, ETL.LABEL_NAME
    FROM DEFINITION_SCHEMA.PG_ELEMENT_TABLE_LABELS AS ETL
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
        ON ( ( ETL.PROPERTY_GRAPH_CATALOG, ETL.PROPERTY_GRAPH_SCHEMA )
           = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
           OR
            S.SCHEMA_OWNER IN ( SELECT ER.ROLE_NAME
                                FROM ENABLED_ROLES AS ER ) )
       AND
          ETL.PROPERTY_GRAPH_CATALOG
        = ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN );

GRANT SELECT ON TABLE PG_ELEMENT_TABLE_LABELS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_ELEMENT_TABLE_LABELS.

15.9 PG_ELEMENT_TABLE_PROPERTIES view

Function

Identify the properties associated with an element table in an SQL-property graph that is owned by a given user or role and is defined in this catalog.

Definition

```
CREATE VIEW PG_ELEMENT_TABLE_PROPERTIES AS
    SELECT ETP.PROPERTY_GRAPH_CATALOG, ETP.PROPERTY_GRAPH_SCHEMA,
           ETP.PROPERTY_GRAPH_NAME, ETP.ELEMENT_TABLE_ALIAS, ETP.PROPERTY_NAME,
           ETP.PROPERTY_EXPRESSION
    FROM DEFINITION_SCHEMA.PG_ELEMENT_TABLE_PROPERTIES AS ETP
      JOIN
          DEFINITION_SCHEMA.SCHEMATA AS S
         ON ( ( ETP.PROPERTY_GRAPH_CATALOG, ETP.PROPERTY_GRAPH_SCHEMA )
            = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
           OR
            S.SCHEMA_OWNER IN ( SELECT ER.ROLE_NAME
                                FROM ENABLED_ROLES AS ER ) )
       AND
          ETP.PROPERTY_GRAPH_CATALOG
        = ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA.CATALOG_NAME AS ISCN );

GRANT SELECT ON TABLE PG_ELEMENT_PROPERTIES
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_ELEMENT_TABLE_PROPERTIES.

15.10 PG_ELEMENT_TABLES view

Function

Identify the element tables in an SQL-property graph that is owned by a given user or role and is defined in this catalog.

Definition

```
CREATE VIEW PG_ELEMENT_TABLES AS
    SELECT ET.PROPERTY_GRAPH_CATALOG, ET.PROPERTY_GRAPH_SCHEMA,
           ET.PROPERTY_GRAPH_NAME, ET.ELEMENT_TABLE_ALIAS, ET.ELEMENT_TABLE_KIND,
           ET.TABLE_CATALOG, ET.TABLE_SCHEMA, ET.TABLE_NAME,
           ET.ELEMENT_TABLE_DEFINITION
    FROM DEFINITION_SCHEMA.PG_ELEMENT_TABLES AS ET
      JOIN
         DEFINITION_SCHEMA.SCHEMATA AS S
        ON ( ( ET.PROPERTY_GRAPH_CATALOG, ET.PROPERTY_GRAPH_SCHEMA )
           = ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
    WHERE ( S.SCHEMA_OWNER = CURRENT_USER
           OR
            S.SCHEMA_OWNER IN ( SELECT ER.ROLE_NAME
                                FROM ENABLED_ROLES AS ER ) )
       AND
          ET.PROPERTY_GRAPH_CATALOG
        = ( SELECT ISCN.CATALOG_NAME
            FROM INFORMATION_SCHEMA_CATALOG_NAME AS ISCN );

GRANT SELECT ON TABLE PG_ELEMENT_TABLES
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_ELEMENT_TABLES.

15.11 PG_LABEL_PROPERTIES view

Function

Identify the properties associated with a label in an SQL-property graph defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PG_LABEL_PROPERTIES AS
    SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
           LABEL_NAME, PROPERTY_NAME
    FROM DEFINITION_SCHEMA.PG_LABEL_PROPERTIES
    WHERE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME )
            IN ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                        PGP.PROPERTY_GRAPH_NAME
                 FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
                 WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                        OR
                         PGP.GRANTEE IN ( SELECT ROLE_NAME
                                          FROM ENABLED_ROLES ) ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_LABEL_PROPERTIES
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_LABEL_PROPERTIES.

15.12 PG_LABELS view

Function

Identify the labels in an SQL-property graph defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PG_LABELS AS
    SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
           LABEL_NAME
    FROM DEFINITION_SCHEMA.PG_LABELS
    WHERE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME )
            IN ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                        PGP.PROPERTY_GRAPH_NAME
                 FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
                 WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                        OR
                         PGP.GRANTEE IN ( SELECT ROLE_NAME
                                          FROM ENABLED_ROLES ) ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_LABELS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_LABELS.

15.13 PG_PROPERTY_DATA_TYPES view

Function

Identify the properties and their data types in an SQL-property graph defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PG_PROPERTY_DATA_TYPES AS
    SELECT PGT.PROPERTY_GRAPH_CATALOG, PGT.PROPERTY_GRAPH_SCHEMA,
           PGT.PROPERTY_GRAPH_NAME, PGT.PROPERTY_NAME, D.DATA_TYPE,
           D.CHARACTER_MAXIMUM_LENGTH, D.CHARACTER_OCTET_LENGTH,
           D.CHARACTER_SET_CATALOG, D.CHARACTER_SET_SCHEMA, D.CHARACTER_SET_NAME,
           D.COLLATION_CATALOG, D.COLLATION_SCHEMA, D.COLLATION_NAME,
           D.NUMERIC_PRECISION, D.NUMERIC_PRECISION_RADIX, D.NUMERIC_SCALE,
           D.DATETIME_PRECISION, D.INTERVAL_TYPE, D.INTERVAL_PRECISION,
           D.USER_DEFINED_TYPE_CATALOG, D.USER_DEFINED_TYPE_SCHEMA,
           D.USER_DEFINED_TYPE_NAME, D.SCOPE_CATALOG, D.SCOPE_SCHEMA, D.SCOPE_NAME,
           D.MAXIMUM_CARDINALITY, D.DECLARED_DATA_TYPE, D.DECLARED_NUMERIC_PRECISION,
           D.DECLARED_NUMERIC_SCALE, D.DTD_IDENTIFIER
    FROM DEFINITION_SCHEMA.PG_PROPERTY_DATA_TYPES AS PGT
      JOIN
         DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D
        ON ( ( PGT.PROPERTY_GRAPH_CATALOG, PGT.PROPERTY_GRAPH_SCHEMA,
               PGT.PROPERTY_GRAPH_NAME, 'PROPERTY GRAPH', PGT.DTD_IDENTIFIER )
           = ( D.OBJECT_CATALOG, D.OBJECT_SCHEMA, D.OBJECT_NAME, D.OBJECT_TYPE,
               D.DTD_IDENTIFIER ) )
    WHERE ( PGT.PROPERTY_GRAPH_CATALOG, PGT.PROPERTY_GRAPH_SCHEMA,
            PGT.PROPERTY_GRAPH_NAME ) IN
            ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                     PGP.PROPERTY_GRAPH_NAME
               FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
               WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                      OR
                       PGP.GRANTEE IN ( SELECT ROLE_NAME
                                        FROM ENABLED_ROLES ) ) )
       AND
          PGT.PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                         FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_PROPERTY_DATA_TYPES
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_PROPERTY_DATA_TYPES.

2) Without Feature T322, “Declared data type attributes”, conforming SQL language shall not reference the following columns in the view INFORMATION_SCHEMA.PG_PROPERTY_DATA_TYPES:

   a) DECLARED_DATA_TYPE.

   b) DECLARED_NUMERIC_PRECISION.

   c) DECLARED_NUMERIC_SCALE.

15.14 PG_PROPERTY_GRAPH_PRIVILEGES view

Function

Identify the privileges on SQL-property graphs defined in this catalog that are available to or granted by a given user or role.

Definition

```
CREATE VIEW PG_PROPERTY_GRAPH_PRIVILEGES AS
    SELECT GRANTOR, GRANTEE, PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
           PROPERTY_GRAPH_NAME, PRIVILEGE_TYPE, IS_GRANTABLE
    FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES
    WHERE ( GRANTEE IN ( 'PUBLIC', CURRENT_USER )
           OR
            GRANTEE IN ( SELECT ROLE_NAME
                         FROM ENABLED_ROLES )
           OR
            GRANTOR = CURRENT_USER
           OR
            GRANTOR IN ( SELECT ROLE_NAME
                         FROM ENABLED_ROLES ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_PROPERTY_GRAPH_PRIVILEGES
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature F231, “Privilege tables”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES.

2) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES.

15.15 PG_VERTEX_DEFINED_LABEL_SETS view

Function

Identify the vertex defined label sets in an SQL-property graph defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PG_VERTEX_DEFINED_LABEL_SETS AS
    SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
           VERTEX_DEFINED_LABEL_SET_IDENTIFIER
    FROM DEFINITION_SCHEMA.PG_VERTEX_DEFINED_LABEL_SETS
    WHERE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME ) IN
            ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                     PGP.PROPERTY_GRAPH_NAME
              FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
              WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                     OR
                      PGP.GRANTEE IN ( SELECT ROLE_NAME
                                       FROM ENABLED_ROLES ) ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PG_VERTEX_DEFINED_LABEL_SETS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PG_VERTEX_DEFINED_LABEL_SETS.

15.16 PROPERTY_GRAPHS view

Function

Identify the SQL-property graphs defined in this catalog that are accessible to a given user or role.

Definition

```
CREATE VIEW PROPERTY_GRAPHS AS
    SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME
    FROM DEFINITION_SCHEMA.PROPERTY_GRAPHS
    WHERE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME ) IN
            ( SELECT PGP.PROPERTY_GRAPH_CATALOG, PGP.PROPERTY_GRAPH_SCHEMA,
                     PGP.PROPERTY_GRAPH_NAME
              FROM DEFINITION_SCHEMA.PG_PROPERTY_GRAPH_PRIVILEGES AS PGP
              WHERE ( PGP.GRANTEE IN ( 'PUBLIC', CURRENT_USER )
                     OR
                      PGP.GRANTEE IN ( SELECT ROLE_NAME
                                       FROM ENABLED_ROLES ) ) )
       AND
          PROPERTY_GRAPH_CATALOG = ( SELECT CATALOG_NAME
                                     FROM INFORMATION_SCHEMA_CATALOG_NAME );

GRANT SELECT ON TABLE PROPERTY_GRAPHS
    TO PUBLIC WITH GRANT OPTION;
```

Conformance Rules

1) Without Feature G850, “SQL/PGQ Information Schema views”, conforming SQL language shall not reference the view INFORMATION_SCHEMA.PROPERTY_GRAPHS.

16 Definition Schema

This Clause modifies Clause 7, “Definition Schema”, in ISO/IEC 9075-11.

16.1 Definition Schema digital artifact

This Subclause modifies Subclause 7.1, “Definition Schema digital artifact”, in ISO/IEC 9075-11.

Insert after the 1st paragraph: These schema definition and manipulation statements are also available from the ISO website as a “digital artifact”. See `https://standards.iso.org/iso-iec/9075/-16/ed-1/en/` to download digital artifacts for this document. To download the schema definition and manipulation statements, select the file named `ISO_IEC_9075-16(E)_PGQ-schema-definition.sql`.

16.2 DATA_TYPE_DESCRIPTOR base table

This Subclause modifies Subclause 7.23, “DATA_TYPE_DESCRIPTOR base table”, in ISO/IEC 9075-11.

Function

The DATA_TYPE_DESCRIPTOR table has one row for each usage of a data type as identified by the ISO/IEC 9075 series. It effectively contains a representation of the data type descriptors.

Definition

Insert into the Definition after

```
  'ROUTINE', 'SEQUENCE'
```

the code:

```
  , 'PROPERTY GRAPH'
```

Description

1) Augment Description 1) by adding “SQL-property graph” as an alternative to the list of objects being described by the values of OBJECT_CATALOG, OBJECT_SCHEMA, and OBJECT_NAME.

2) Augment Description 1) by adding “'PROPERTY GRAPH'” to the list for OBJECT_TYPE.

Initial Table Population

No additional Initial Table Population items.

16.3 TABLES base table

This Subclause modifies Subclause 7.55, "TABLES base table", in ISO/IEC 9075-11.

Function

The TABLES table contains one row for each table including views. It effectively contains a representation of the table descriptors.

Definition

Insert into the Definition after

```
    );
```

the code:

```
ALTER TABLE TABLES ADD
  CONSTRAINT TABLES_CHECK_NOT_PROPERTY_GRAPH
    CHECK ( NOT EXISTS (
              SELECT PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                     PROPERTY_GRAPH_NAME
               FROM PROPERTY_GRAPHS
               WHERE ( TABLE_CATALOG, TABLE_SCHEMA, TABLE_NAME )
                   = ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                       PROPERTY_GRAPH_NAME ) );
```

Description

No additional Descriptions.

Initial Table Population

No additional Initial Table Population items.

16.4 PG_DEFINED_LABEL_SETS base table

Function

The PG_DEFINED_LABEL_SETS base table has one row for each defined label set in a pure property graph descriptor included in an SQL-property graph descriptor.

Definition

```
CREATE TABLE PG_DEFINED_LABEL_SETS (
    PROPERTY_GRAPH_CATALOG                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PROPERTY_GRAPH_SCHEMA                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PROPERTY_GRAPH_NAME                         INFORMATION_SCHEMA.SQL_IDENTIFIER,
    DEFINED_LABEL_SET_IDENTIFIER                INFORMATION_SCHEMA.SQL_IDENTIFIER,

    CONSTRAINT PG_DEFINED_LABEL_SETS_PRIMARY_KEY
      PRIMARY KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, DEFINED_LABEL_SET_IDENTIFIER ),

    CONSTRAINT PG_DEFINED_LABEL_SETS_FOREIGN_KEY_PROPERTY_GRAPHS
      FOREIGN KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME )
        REFERENCES PROPERTY_GRAPHS
    );
```

Description

1) The values of PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, and PROPERTY_GRAPH_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the SQL-property graph that includes the defined label set identified by the value of DEFINED_LABEL_SET_IDENTIFIER.

2) The value of DEFINED_LABEL_SET_IDENTIFIER is an implementation-dependent (UV025) value that uniquely identifies the defined label set being described among all defined label sets of the SQL-property graph identified by PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, and PROPERTY_GRAPH_NAME.

Initial Table Population

None.

16.5 PG_DEFINED_LABEL_SET_LABELS base table

Function

The PG_DEFINED_LABEL_SET_LABELS base table has one row for each label included in a defined label set in a pure property graph descriptor included in an SQL-property graph descriptor.

Definition

```
CREATE TABLE PG_DEFINED_LABEL_SET_LABELS (
    PROPERTY_GRAPH_CATALOG                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PROPERTY_GRAPH_SCHEMA                        INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PROPERTY_GRAPH_NAME                          INFORMATION_SCHEMA.SQL_IDENTIFIER,
    DEFINED_LABEL_SET_IDENTIFIER                 INFORMATION_SCHEMA.SQL_IDENTIFIER,
    LABEL_NAME                                   INFORMATION_SCHEMA.SQL_IDENTIFIER,

    CONSTRAINT PG_DEFINED_LABEL_SET_LABELS_PRIMARY_KEY
      PRIMARY KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, DEFINED_LABEL_SET_IDENTIFIER, LABEL_NAME ),

    CONSTRAINT PG_DEFINED_LABEL_SET_LABELS_FOREIGN_KEY_PG_DEFINED_LABEL_SETS
      FOREIGN KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, DEFINED_LABEL_SET_IDENTIFIER )
        REFERENCES PG_DEFINED_LABEL_SETS,

    CONSTRAINT PG_DEFINED_LABEL_SET_LABELS_FOREIGN_KEY_PG_LABELS
      FOREIGN KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, LABEL_NAME )
        REFERENCES PG_LABELS
    );
```

Description

1) The values of PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, and PROPERTY_GRAPH_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the SQL-property graph that includes the defined label set identified by the value of DEFINED_LABEL_SET_IDENTIFIER.

2) The value of LABEL_NAME is the name of the label being described that is part of the defined label set identified by DEFINED_LABEL_SET_IDENTIFIER.

Initial Table Population

None.

16.6 PG_EDGE_DEFINED_LABEL_SETS base table

Function

The PG_EDGE_DEFINED_LABEL_SETS base table has one row for each edge label set in a pure property graph descriptor included in an SQL-property graph descriptor.

Definition

```
CREATE TABLE PG_EDGE_DEFINED_LABEL_SETS (
    PROPERTY_GRAPH_CATALOG                        INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PROPERTY_GRAPH_SCHEMA                         INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PROPERTY_GRAPH_NAME                           INFORMATION_SCHEMA.SQL_IDENTIFIER,
    EDGE_DEFINED_LABEL_SET_IDENTIFIER             INFORMATION_SCHEMA.SQL_IDENTIFIER,
    IS_DIRECTED                                   INFORMATION_SCHEMA.YES_OR_NO
      CONSTRAINT PG_EDGE_DEFINED_LABEL_SETS_IS_DIRECTED_NOT_NULL
        NOT NULL,
    IS_UNDIRECTED                                 INFORMATION_SCHEMA.YES_OR_NO
      CONSTRAINT PG_EDGE_DEFINED_LABEL_SETS_IS_UNDIRECTED_NOT_NULL
        NOT NULL,

    CONSTRAINT PG_EDGE_DEFINED_LABEL_SETS_PRIMARY_KEY
      PRIMARY KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, EDGE_DEFINED_LABEL_SET_IDENTIFIER ),

    CONSTRAINT PG_EDGE_DEFINED_LABEL_SETS_FOREIGN_KEY_PG_DEFINED_LABEL_SETS
      FOREIGN KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, EDGE_DEFINED_LABEL_SET_IDENTIFIER )
        REFERENCES PG_DEFINED_LABEL_SETS,

    CONSTRAINT PG_EDGE_DEFINED_LABEL_SETS_CHECK_DIRECTED_UNDIRECTED
      CHECK ( NOT ( IS_DIRECTED = 'NO'
                 AND
                    IS_UNDIRECTED = 'NO' ) )
    );
```

Description

1) The values of PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME, and EDGE_DEFINED_LABEL_SET_IDENTIFIER are the values of PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME, and DEFINED_LABEL_SET_IDENTIFIER, respectively, of the row in PG_DEFINED_LABEL_SETS that describes the edge defined label set.

2) The values of IS_DIRECTED and IS_UNDIRECTED have the following meanings:

   Case:

   a) If the value of the IS_DIRECTED column is 'YES' and the value of the IS_UNDIRECTED column is 'YES', then an edge that has the edge defined label set being described is possibly directed or undirected.

   b) If the value of the IS_DIRECTED column is 'YES', then an edge that has the edge defined label set being described is always directed.

   c) If the value of the IS_UNDIRECTED column is 'YES', then an edge that has the edge defined label set being described is always undirected.

Initial Table Population

None.

16.7 PG_EDGE_TABLE_COMPONENTS base table

Function

The PG_EDGE_TABLE_COMPONENTS base table has one row for each pair of columns that is part of the edge destination key/destination vertex key or edge source key/source vertex key of an edge table included in a tabular property graph descriptor included in an SQL-property graph descriptor.

Definition

```
CREATE TABLE PG_EDGE_TABLE_COMPONENTS (
    PROPERTY_GRAPH_CATALOG                      INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PROPERTY_GRAPH_SCHEMA                       INFORMATION_SCHEMA.SQL_IDENTIFIER,
    PROPERTY_GRAPH_NAME                         INFORMATION_SCHEMA.SQL_IDENTIFIER,
    EDGE_TABLE_ALIAS                            INFORMATION_SCHEMA.SQL_IDENTIFIER,
    VERTEX_TABLE_ALIAS                          INFORMATION_SCHEMA.SQL_IDENTIFIER,
    EDGE_END                                    INFORMATION_SCHEMA.CHARACTER_DATA
      CONSTRAINT PG_EDGE_TABLE_COMPONENTS_CHECK_EDGE_END
        CHECK ( EDGE_END IN ( 'DESTINATION', 'SOURCE' ) ),
    EDGE_TABLE_COLUMN_NAME                      INFORMATION_SCHEMA.SQL_IDENTIFIER
      CONSTRAINT PG_EDGE_TABLE_COMPONENTS_EDGE_TABLE_COLUMN_NAME_NOT_NULL
        NOT NULL,
    VERTEX_TABLE_COLUMN_NAME                    INFORMATION_SCHEMA.SQL_IDENTIFIER
      CONSTRAINT PG_EDGE_TABLE_COMPONENTS_VERTEX_TABLE_COLUMN_NAME_NOT_NULL
        NOT NULL,
    ORDINAL_POSITION                            INFORMATION_SCHEMA.CARDINAL_NUMBER
      CONSTRAINT PG_EDGE_TABLE_COMPONENTS_CHECK_ORDINAL_POSITION_GREATER_THAN_ZERO
        CHECK ( ORDINAL_POSITION > 0 )
      CONSTRAINT PG_EDGE_TABLE_COMPONENTS_CHECK_ORDINAL_POSITION_CONTIGUOUS
        CHECK ( 0 = ALL ( SELECT MAX(ORDINAL_POSITION) - COUNT(*)
                          FROM PG_EDGE_TABLE_COMPONENTS
                          GROUP BY PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                                   PROPERTY_GRAPH_NAME, EDGE_TABLE_ALIAS ) ),

    CONSTRAINT PG_EDGE_TABLE_COMPONENTS_PRIMARY_KEY
      PRIMARY KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, EDGE_TABLE_ALIAS, VERTEX_TABLE_ALIAS,
                    EDGE_END, ORDINAL_POSITION),

    CONSTRAINT PG_EDGE_TABLE_COMPONENTS_FOREIGN_KEY_EDGE_IN_PG_ELEMENT_TABLES
      FOREIGN KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, EDGE_TABLE_ALIAS )
        REFERENCES PG_ELEMENT_TABLES,

    CONSTRAINT PG_EDGE_TABLE_COMPONENTS_FOREIGN_KEY_VERTEX_IN_PG_ELEMENT_TABLES
      FOREIGN KEY ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA,
                    PROPERTY_GRAPH_NAME, VERTEX_TABLE_ALIAS )
        REFERENCES PG_ELEMENT_TABLES,

    CONSTRAINT PG_EDGE_TABLE_COMPONENTS_UNIQUE_EDGE_COLUMN
      UNIQUE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
               EDGE_TABLE_ALIAS, VERTEX_TABLE_ALIAS, EDGE_TABLE_COLUMN_NAME ),

    CONSTRAINT PG_EDGE_TABLE_COMPONENTS_UNIQUE_VERTEX_COLUMN
      UNIQUE ( PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, PROPERTY_GRAPH_NAME,
               EDGE_TABLE_ALIAS, VERTEX_TABLE_ALIAS, VERTEX_TABLE_COLUMN_NAME )
    );
```

Description

1) The values of PROPERTY_GRAPH_CATALOG, PROPERTY_GRAPH_SCHEMA, and PROPERTY_GRAPH_NAME are the catalog name, unqualified schema name, and qualified identifier, respectively, of the SQL-property graph that includes the edge table *ET* identified by the value of EDGE_TABLE_ALIAS and the vertex table *VT* identified by VERTEX_TABLE_ALIAS.

2) The value of EDGE_END is

   Case:

   a) If *VT* is the destination vertex table of *ET*, then 'DESTINATION'.

   b) Otherwise, (*VT* is the source vertex of *ET*), 'SOURCE'.

3) The value of EDGE_TABLE_COLUMN_NAME is

   Case:

   a) If *VT* is the destination vertex table of *ET*, then the name of the column that is part of the edge destination key.

   b) Otherwise, (*VT* is the source vertex of *ET*), the name of the column that is part of the edge source key.

4) The value of VERTEX_TABLE_COLUMN_NAME is

   Case:

   a) If *VT* is the destination vertex table of *ET*, then the name of the column that is part of the destination vertex key.

   b) Otherwise, (*VT* is the source vertex of *ET*), the name of the column that is part of the source vertex key.

5) The value of ORDINAL_POSITION is the ordinal position of the specific columns in the edge destination key/destination vertex key or edge source key/source vertex key, respectively.

Initial Table Population

None.