Property Graph Data Model for SQL

Title	Property Graph Data Model for SQL
Authors	Neo4j SQL working group[1]
Status	Outline partial draft of initial working document on SQL PGQ
Date	24 April 2018
Date of r1	24 April 2018 Sub-editorial changes
Date of r2	2 May 2018 Correct over-restriction on sharing of property names in Label Sets, strengthens References and comparison of PGDM model variants.

Contents

[1] Current members of the Neo4j SQL working group are: Tobias Lindaaker, Stefan Plantikow, Petra Selmer, Peter Furniss, Alastair Green.

1. Summary

We present a revised version of the section **“SQL Property Graph Data Model”** in **[sql-pg-2017-0029]**.

The meta-metamodel of such a Graph Schema and of a Graph Object, as new entities in the SQL Information Schema, is shown in UML.

This contribution forms the basis for a second (forthcoming) paper on mapping existing tabular data in a SQL store into a Graph Object.

Readers may also find the graph metamodel representation proposed in **[openCypher-CIR-2018-307]** interesting and useful in understanding the model presented in this paper.

That “ASCII Art” representation is another view of the kind of **Graph Schema** or **Property Graph Data Model** defined here**.** It reflects a separate proposal documented in **[openCypher-CIR-2018-311]**, which allows use of Cypher patterns in SQL without the need for syntax “quoting”.)

The ASCII Art schema representation allows the definition of Label Sets, which is an important step in forming a Graph Schema.

2. References

[Foundation:2016]	Jim Melton (ed), “ISO International Standard (IS) Database Language SQL- Part 2: SQL/Foundation”, ISO/IEC 9075-2:2016
[sql-pg-2017-0029]	Peter Furniss, Alastair Green, “SQL Property Graph Representations”, July 2017
[openCypher-CIR-2018-307]	Neo4j Cypher Language Group, “‘ASCII Art’ graph schema”, March 2018
[sql-pg-2017-0036]	Alastair Green, “Remarks on [sql-pg-2017-0032r1] ‘Comments on sql-pg-2017-0029’”, July 2017
[sql-pg-2017-0047r1]	Peter Furniss, Alastair Green, Petra Selmer, “SQL Graph Query Procedures” (with corrigenda), August/October 2017
[openCypher-CIR-2018-311]	Neo4j SQL Working Group, “Syntax modifications to Cypher path patterns to permit use in SQL”, April 2018
[Cypher 9]	Neo4j Cypher Language Group, “Cypher Query Language Reference, Version 9”, 2018
[PGQL 1.1]	Oracle, “PGQL 1.1 Specification”, 2018
[GCORE]	Angles et al., “G-CORE: A Core for Future Graph Query Languages”, pre-print of paper accepted for SIGMOD 2018

3. Purpose of a Property Graph Object in SQL

The purpose of a property graph object is to provide an operand or a result of a property graph function. Such a function, however surfaced in SQL syntax, is an operation that accepts as input one or more graphs.

In [sql-pg-2017-0047r1] we proposed one such function, GRAPH_TABLE, that can be used to illustrate this point.

> Drafting Note
>
> *We also proposed that graph objects be exposed as cell values with a type called GRAPH_REFERENCE.*
>
> *We have subsequently come to the view that the simplest way to create a reference to a graph is to create a named Information Schema object, analogous to a table. (We referred to this possibility in Appendix B, p30 of the cited document, where we also suggested the syntactic impact that might have on GRAPH_TABLE.)*
>
> *In the interests of making progress on initial property graph extensions to SQL we are focussing on the option of named graph objects, and that is reflected in this paper.*

4. The SQL Property Graph Data Model

Simplifying, constraining and making fully logical the data model defined in the section **"SQL Property Graph Data Model"** in [sql-pg-2017-0029], we define a **Property Graph Data Model** for SQL as follows.

1) A **Property Graph** is a directed multigraph[2], whose vertices or nodes form a set of **Nodes**, and whose edges or arcs or relationships form a set of **Edges**. The Nodes and Edges of a graph are, taken together, the **Entities** of the graph.
2) Each Edge has one **Start Node** and one **End Node**.
3) The Direction of an Edge is from Start Node to End Node: synonymously, the Start Node is the tail and the End Node is the head.
4) The term **Graph** is used hereafter as a synonym for Property Graph. A Graph is more than a directed multigraph in the following respects.
5) A Graph has a **Name,** which is an identifier that follows the rules for naming user-defined Tables. A Graph Name can be qualified by the database and schema in which the Graph is defined, giving a triple-element name like that of a Table.
6) Every Entity has one to many **Labels** (which are string identifiers). An implementation must define the maximum number of Labels allowed for Nodes and for Edges, which in each case may be a finite positive integer, or infinity.
7) Each Label can have zero to many **Properties** associated with it.
8) An Entity cannot have two Labels each of which has a Property of the same name, unless each of those properties has the same data-type[3].
9) A **Property** is a named typed value, whose type is any valid SQL type, and which may be optional or mandatory for any given Node or Edge.
10) A **Non-Null Property** is an optional property which has a null value.
11) A **Property Name** must be unique within the set of Property Names associated with a Label.
12) A Label can have one or more **Entity Keys**. An **Entity Key** is a set of mandatory Properties whose values, taken together, form a set, an **Entity Key Value**.
13) If a Node has a Label, and that Label has an Entity Key, then that Entity Key acts as a **Node Key**. The Entity Key Value of a Node Key for a given Node cannot equal the

[2] A *multigraph* is one where there can be multiple edges between two nodes and where an edge can connect a node to itself (a *loop*). Every edge in a *directed graph* has a tail and a head, and the direction flows from the tail to the head.

[3] For this to be useful, the properties should have the same application semantic.

Entity Key Value of any other Node with the same Label[4]. An implementation may choose not to support Node Keys.

14) If an Edge has a Label, and that Label has an Entity Key, then that Entity Key acts as an **Edge Key**. The Entity Key Value of an Edge Key for a given Edge cannot equal the Entity Key Value of any other Edge with the same Label. An implementation may choose not to supports Edge Keys.

15) If an Entity has more than one Label then all of its Labels are called a **Label Set**. The union of all of the Properties of all the members of a Label Set is a set called a **Label Set Properties**. It follows that each Property of such an Entity has a name which is unique. To achieve this, if two or more Labels in the Labet Set have a property of the same name, then (if the data-type of each such property is identical) the Label Set Properties will have only one property with that name[5].

16) The set of Non-Null Properties which is a subset of the Label Set Properties of an Entity (of a Node, or of an Edge) is a set called the **Non-Null Properties**. The set of values of all the members of the Properties Present is the **Non-Null Properties Values**.

17) For any two Nodes which have the same Label, the corresponding Non-Null Properties may be equal or different. If equal, then the corresponding Non-Null Properties Values for the same two Nodes may be equal or different. The set of Nodes in a Graph having a given Label therefore maps to a multiset of **Node Non-Null Properties** and to a multiset of **Node Non-Null Properties Values**.

18) For any two Edges which have the same Label, the corresponding Non-Null Properties may be equal or different. If equal, then the corresponding Non-Null Properties Values for the same two Edges may be equal or different. The set of Edges in a Graph having a given Label therefore maps to a multiset of **Edge Non-Null Properties** and to a multiset of **Edge Non-Null Properties Values**.

[4] Node and Edge Keys are candidate or unique keys. The concepts of primary key and foreign key to do not exist in the property graph data model, where relationships are expressed explicitly by edges. They can be relevant in assisting *mappings* between the relational and property graph data models.

[5] The coalescence of properties that have the same meaning in more than one Label is equivalent to the effect of creating a “mix-in” interface in Java, to take an analogy.

5. Remarks on the Property Graph Data Model

5.1. Graph Entities without Properties

Graphs, which can express structure without data values, may have no properties on a particular kind of Edge or Node. This is common and unremarked for Edges, and less common for Nodes, but Nodes which are linked to form purely structural models are found in the wild.

A Table in SQL must have at least one column. This extends from base tables to views (given the same restriction on the results of SELECT).

Labels, for this reason are logically distinct from Tables, and cannot always be represented by a Table, unless the model (artificially) forces at least one property to be added to every Label.

A View in SQL cannot have a primary key or uniqueness constraint. Therefore, not all Tables can have unique keys. Labels should have the ability to have candidate keys (unique keys). This is a second reason for distinguishing Labels from Tables.

Therefore, it is, in our view, preferable to express a Graph Object as being conformant with a Graph Schema, which implements a metamodel consistent with the overall Data Model (schema metamodel) described here, and to express optional mappings from Tables to Labels to give a Graph view over existing data, than to force the use of Tables as the building blocks of a Graph Schema, not least because this would preclude the creation of Graph Objects that do not derive from nor are directly mapped to Table data.

5.2. Explicit Entity Identifiers Removed

Node identifiers and Edge Identifiers (Entity Identifiers) have been removed in this revised version of the data model. Their presence in **[sql-pg-2017-0029]** was a mistake, influenced by implementation tactics in existing SQL property graph implementations like SQLServer.

[Cypher 9] also contributes to this erroneous thinking by making a function available which gives a unique edge identifier or node identifier for an entity accessible via a binding variable, and by defining the identity of two node references in terms of identifier equality.

[PGQL 1.1] does not define identifiers as an inherent part of the data model. Conversely, **[GCORE]** makes identifiers central to its data model.

However, at a logical level, each Node and each Edge is a member of a set (like each row in a table is a member of a set of rows), and is therefore unique, whatever the number or value of its properties. Two entities with identical Label Sets and identical values of the Label Set Properties are still two distinct entities, just as two rows in a Table are distinct tuples.

At the same time the same two entities, viewed in terms of their Label Sets and values of their Label Set Properties, form a multiset. SQL uses this angle of view when it describes the data content of a Table as a "multiset of rows" **[Foundation:2016]**.

It is not necessary for the specification of graph query semantics, or for the definition of Labels, or for the definition of the data model in terms of structural relationships (edges between nodes), to use the concept of Entity Identifier. It is only necessary to define a function, that may or may not be available in the syntax of any graph manipulation sub-language, that allows for the comparison of two Entity references and evaluates to either "identical: the same Entity", or "not identical: different Entities".

Row identifiers are not part of the core SQL tabular data model, for the same reasons.

5.3. Closed-world Schema

Note that the data model defined above

a) prevents data being held in a property graph in Properties whose type is not defined by the Information Schema

b) prevents data in a Property for a given Label from having more than one type.

These constraints corresponds to the fact that SQL Information Schema is a closed-world model, which extends (modulo Polymorphic Table Functions) to column-typing. This allows schema-based type inference during the course of projection or expression evaluation.

Unlike Cypher, SQL property graphs will have mandatory schema (will be "schema strict"), and will not permit heterogeneous property typing.

Cypher's data model is "schema optional" and does allow types of a given property name to vary. The SQL Property Graph Data Model defined here is therefore a proper subset of Cypher's[6]. In particular a Cypher read or data update query can operate unchanged over a graph conforming to this SQL Property Graph Data Model.

5.4. Label Sets and Inheritance

If the relationship between a type (class) of Start Node, a type of Edge and a type of End Nodes is defined in terms of Label Sets (i.e., where the Label Set is the type), then it is possible to say things like: "All Persons are either Residents or Visitors, for the purpose of a census; Any Person may be present in a Town, on the night of the census."

[6] There is a proviso: openCypher has not yet formalized the concepts of Node Keys and Relationship Keys. Node Keys are present in Neo4j Cypher. Neo4j favours introducing both kinds of Entity Key in Cypher, but that depends on a more general expansion of schema features in Cypher.

It therefore follows that the relationship or edge **`[PRESENT_IN]`** can connect nodes of type **`(Person)`** and **`(Town)`**, and that it is not possible to be a **`(Person)`** without being a **`(Resident | Visitor)`**. This allows the semantic effect of a supertype, like **`(Person)`**, with respect to to a subtype, like **`(Resident)`** or **`(Visitor)`**, to be expressed without explicitly describing it as an inheritance relationship.

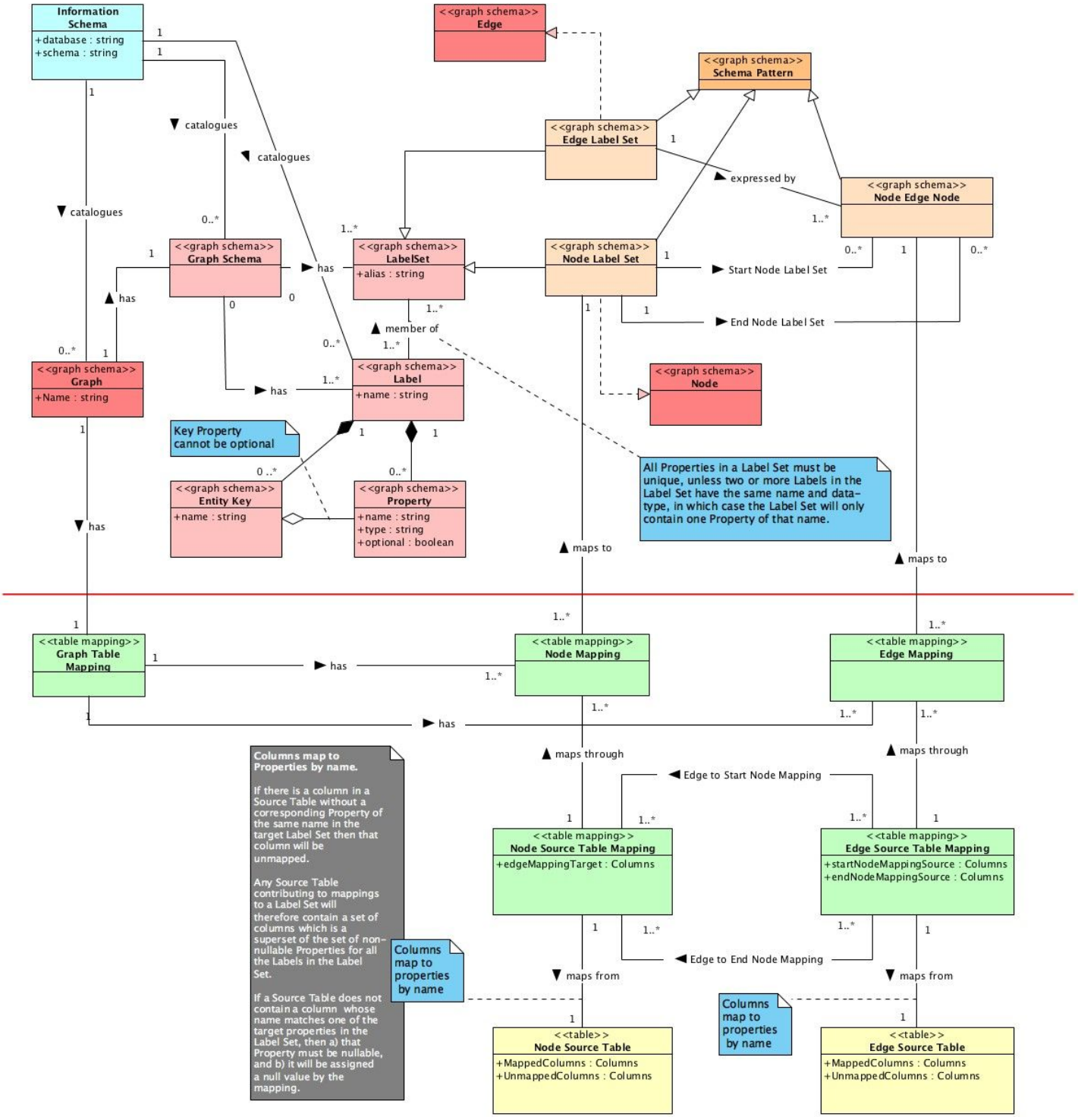
An “accidental”, coincidental combination of labels, where some nodes of type **`(Audited)`** are also nodes of type **`(Person)`**, but not all nodes of type **`(Person)`** are **`(Audited)`**, can also be expressed in this model.

This is achieved by defining Label Sets and defining “Node Edge Node” (NEN) relationships in terms of those Label Sets.

[openCypher-CIR-2018-307] is a means of stating those definitions and NEN relationships (*mutatis mutandis* for the likely differences in Cypher and SQL DDL syntax).

The ability to state these Graph Schema constraints without regard to mappings from Tables to Label Sets is useful in factoring different concerns in the design and programming phases of creating Graph Objects.

6. UML Graph Schema Metamodel (Property Graph Data Model)



7. An ITI and openCypher contribution from Neo4j Inc.

This contribution is a Deliverable under the terms of clause 2.2.1 of the Agreement for Membership in the InterNational Committee for Information Technology Standards ("INCITS"), a Division of the Information Technology Industry Council ("ITI") to which Neo4j Inc. is a party.

It is also a contribution to the openCypher community[7] and like all such contributions is:

This document is a contribution by Neo4j's SQL working group to the openCypher project and to the SQL standards formation process.

[7] https://www.opencypher.org/