# Aligning Cypher and SQL Graph Pattern Syntax

Title Aligning Cypher and SQL Graph Pattern Syntax
Author Neo4j SQL working group[1]
Date 26 April 2018

This paper was originally submitted to the openCypher community process as an annexe to a Cypher Improvement Request (CIR) [CIR-2018-311], on 23 April 2018.

In that form it was titled SQL-compatible Expanded Graph Pattern Language and was presented as a "Preliminary CIP for CIR-2018-311". A CIP is a Cypher Improvement Proposal, a proposed solution to the issues posed in the original CIR.

We are submitting the same content to the INCITS SQL Property Graph Ad-Hoc as a contribution to the proposed Property Graph Query extensions to SQL. The goal of this proposal is threefold:

1) To converge Cypher, PGQL and SQL graph pattern syntax, to assist users who wish to operate across those languages
2) To enable the use of one variant of that graph pattern syntax in SQL `MATCH` clauses, without clashing with existing SQL syntactic constructs that have established semantics
3) To enable the use of a Cypher query within a SQL graph function such as `GRAPH_TABLE`, allowing Cypher queries to be composed as SQL sub-queries without the need for "quoting".

Part of this proposal (e.g. use of `IS` to mark label names) reflects a similar approach first used in the Ad Hoc's work in [sql-pg-2018-0001]. There is an additional issue raised by the same paper, which is the syntax for expressing conjunction, alternation or (potentially) negation of a label in a node or edge component of a pattern. This paper does not address that. Currently Cypher expresses conjunction by concatenating labels within the node parentheses, does not support multiple edge labels, and expresses alternation between single labels for edges (but not nodes). It is clear that a consistent set of logical operations on labels for nodes

[1] The Neo4j SQL working group currently has the following members: Petra Selmer, Stefan Plantikow, Tobias Lindaaker, Alastair Green, Peter Furniss.

and edges would be desirable. In the spirit of this paper we would advocate that the existing syntax of Cypher e.g. `(:Person :Developer)` in this respect be preserved as a permitted syntactic option.

---

*This draft CIP aims to resolve CIR-2018-311. This document will be converted to ASCII Doc and upgraded to the normal standard of completeness of an openCypher CIP (grammar changes described formally, survey of related features in other languages etc), but we'd like to get the essence of this proposal documented. This will facilitate preparation for the forthcoming Cypher Implementers Group meeting on 22-24 May in Copenhagen, and for discussions in the SQL standards process.*

## References


| | |
|---|---|
| [sql-pg-2018-0001] | Fred Zemke, “GRAPH_TABLE”, ANSI INCITS sql-pg-2018-0001, 9 April 2018 |
| [CIR-2018-311] | Cypher Improvement Request, “Syntax modifications to Cypher path patterns to permit use in SQL” |
| [Cypher 9] | Cypher Query Language Reference, Version 9 |
| [Cypher SIGMOD18] | Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andres Taylor. SIGMOD 2018. *“Cypher: An Evolving Query Language for Property Graphs”* |


## 1. Introduction

The Neo4j SQL working group has been thinking about ways of allowing Cypher queries to be included in SQL queries without “quoting”.

The motive for this is to allow a common core of graph querying features to be defined, which is shared by Cypher and SQL. We’d also like to share that core with PGQL or any other proposed property graph query language.

A common core (intersection of SQL, Cypher and other languages) will help convergence on a standard for property graph querying, which is the overall aim of the openCypher project.

The leading thought is that Cypher pattern syntax will be extended to allow some additive variations on existing Cypher syntax, which would

a) be options in Cypher 10 (see [Cypher SIGMOD18])
b) allow existing Cypher 9 syntax to operate unaltered (see [Cypher 9])
c) optional or mandatory SQL syntax, depending on the characteristics of the SQL implementation.

## 2. Problems

[CIR-2018-311] summarizes the problems: if a Cypher pattern, or a pattern whose syntax is shared between Cypher and SQL and PGQL, is to be used in SQL then it must deal with these three obstacles (which have been raised by participants in the INCITS Ad Hoc on Property Graph Extensions (for SQL):

1) The colon string is used to introduce a host variable (parameter) in standard SQL. While the use of colon in graph patterns could be considered to be sufficiently contextualized to avoid misinterpretation as a host variable introducer, it is quite likely that SQL implementations will treat colon as being a “lexing exit”, permitting the introduction of a substitute value prior to “true lexing and parsing”. (Some uses of colon allow the string to be used to introduce and terminate a keyword, but this is considered to be easily detectable by a “cheap pre-processor”.)

2) Square brackets are used in Microsoft SQLServer and other Microsoft SQL dialects to delimit identifiers for objects like tables. Although this is not provided for in the SQL standard, it is such a pervasive feature of user queries written for one of the largest suites of SQL-based products in production use, that it should be treated as de facto conformance option of ISO SQL.

3) The double-hyphen string introduces a quote in SQL. Comments are removed as part of lexical processing, and it is therefore impossible for this comment introducer to be used as a syntactic element with semantic import in SQL.

## 3. Proposed Solutions

### 3.1. Labeled Node Patterns

- Node patterns are always enclosed in parentheses.

- A new variant form of node pattern allows the colon used in existing Cypher and PGQL patterns to specify the label of a node to be optionally replaced by the reserved word IS, giving a variant syntactic form `<variable> IS <label>`.

- As in Cypher today, both the variable and the label specifier are optional.
    - If the variable is omitted, a fresh variable name is generated implicitly.
    - If the label specifier is omitted, every node, irrespective of label, is matched by the node pattern.

The proposed syntax is summarized in the following table. The column headed “Optional Variant” shows the new syntax that equates to the existing syntax shown in the column headed “Current Cypher/PGQL”. Both forms would be allowed in Cypher.

SQL would be free to use either form, and the use of the existing form could be treated in the SQL standard as an optional “higher conformance feature”.

| **Purpose** | **Optional Variant** | **Current Cypher/PGQL** |
|---|---|---|
| Node pattern with variable name and label | `(p IS Person)` | `(p:Person)` |
| Node pattern with variable name only (no label) | `(p)` | `(p)` |
| Node pattern with label only (no variable name) | `(IS Person)` | `(:Person)` |
| Bare node pattern (having neither variable name nor label) | `()` | `()` |

### 3.2. Labeled Edge/Relationship Patterns

- Variables and labels are specified using the same syntactic forms as proposed for node patterns.
- Square brackets are made optional.
- Two columns are used to show the (slight) variation between existing Cypher 9 and PGQL.

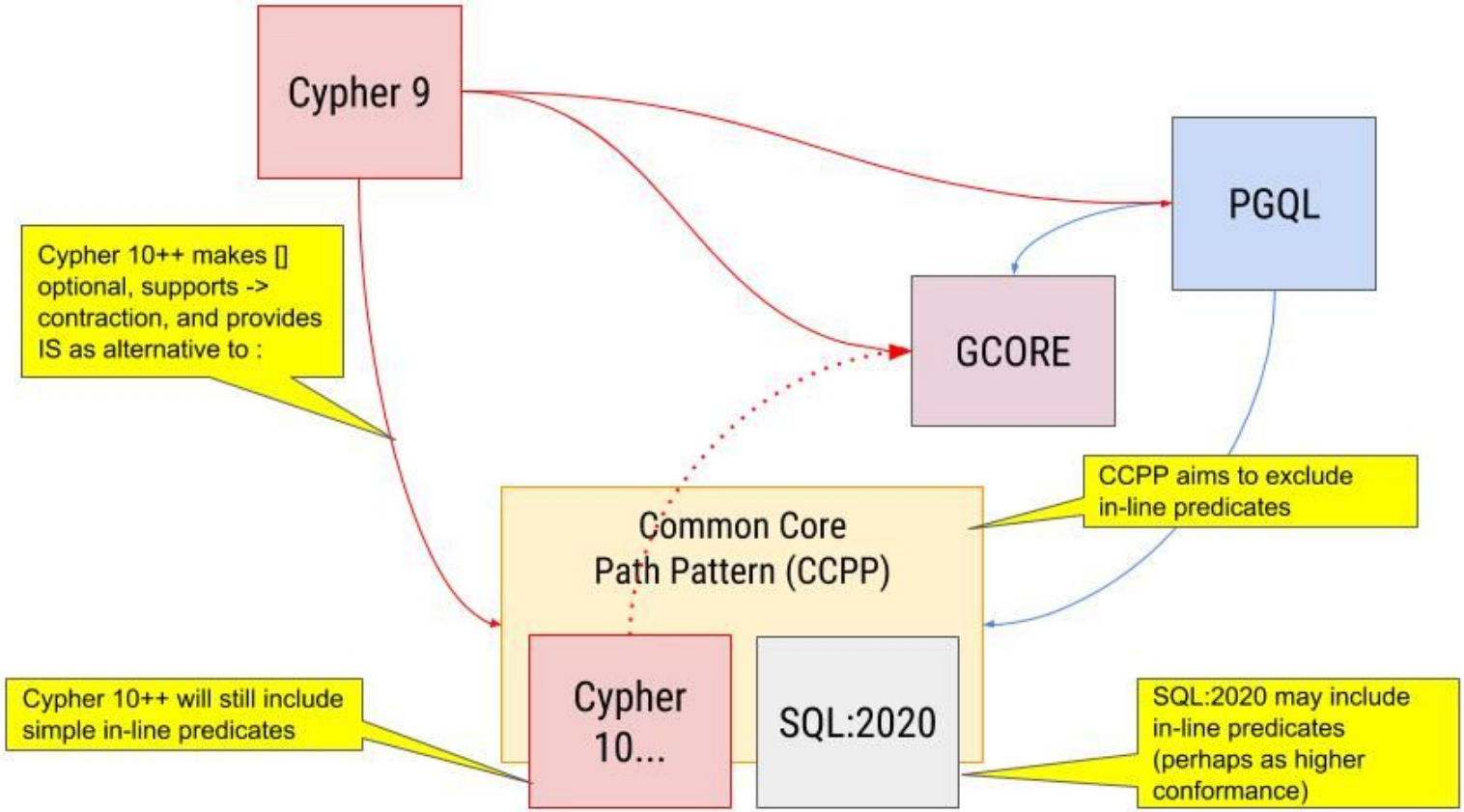
| **Purpose** | **Optional Variants** | **Cypher 9** | **PGQL** |
|---|---|---|---|
| Edge pattern with variable name and label | `-[e IS LIKES]->`<br>`- e IS LIKES ->`<br>`- e:LIKES ->` | `-[e:LIKES]->` | `-[e:LIKES]->` |
| Edge pattern with variable name only (no label) | `-[e]->`<br>`- e ->` | `-[e]->` | `-[e]->` |
| Edge pattern with label only (no variable name) | `-[IS LIKES]->`<br>`- IS LIKES ->`<br>`- :LIKES ->` | `-[:LIKES]->` | `-[:LIKES]->` |
| Bare edge pattern (having neither a variable name nor label) | `-[]->`<br>`->` | `-[]->`<br>`-->` | `-[]->`<br>`-->` (1.0)<br>`->` (1.0 / 1.1) |

### 3.3. Shorthand for Edge/Relationship Pattern Direction where no Relationship is Specified

| Purpose | Using Brackets | Using No Brackets | Cypher 9 Shorthand | Optional Variant Shorthand |
|---|---|---|---|---|
| Directed outgoing edge | `()-[e]->()` | `()- e ->()` | `()-->()` | `()->()` |
| Directed incoming edge | `()<-[e]-()` | `()<- e -()` | `()<--()` | `()<-()` |
| Any edge regardless of direction | `()-[e]-()` | `()- e -()` | `()--()`<br>`()<-->()` | `()-()`<br>`()<->()` |

## 4. Appendix: Graph Pattern Syntax Family Tree

This diagram depicts the history of how graph pattern syntax has evolved over time across property graph query languages:



## 5. 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 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.**