Comparing PGQL,G-Core, Cypher

Presentation at oCIM 4 on May 22-24, 2018

Stefan Plantikow (Neo4j)

Paper WG3 YTZ-030r1

Summary Chart of Cypher, PGQL, and G-Core

Title Summary Chart of Cypher, PGQL, and G-Core
Authors Stefan Plantikow, U.S. National Expert
Date 2018-05-14

Contents

1. Introduction **2**

2. References **3**

3. Language features **4**

3.1. Shared Core 4

3.2. Language design scope 5

3.3. Query structure and clause order 5

Cypher

- Original (1st) declarative query language for *property graphs*
- "Ascii-Art" pattern syntax for graph matching
- SQL-like syntax
- Full DML
- DDL for constraints and indices
- Isomorphism by default
- Main language in actual use, 10 implementations, 5 products

- Planned: Multiple graphs, query composition, path patterns

PGQL

- Oracle's take on property graphs
- Pattern matching with "Ascii-Art"
- Path patterns / RPQs with macros
- Homomorphism by default
- Limited multiple graphs support
- 2 implementations, 1 product

G-Core

- LDBC's take on property graphs
- Core language only
- Pattern matching with "Ascii-Art"
- Path patterns / RPQs with macros
- Homomorphism by default
- Multiple graphs support
- Graph construction and graph query composition
- 1 research implementation

Shared Core

I Pattern matching using "Ascii-Art"

II Returns table or graph

III SQL-like syntax

IV Sequential flow with some binary operators (like UNION)

Shared Core

Cypher 9	`FROM languageGraph`
	`MATCH (a:Engineer)-[:LIKES]->(l:Language)`
	`WHERE (l)-[:SUPPORTS]->(:Feature {name: "Pattern Matching"})`
	`RETURN a.name, l.name`

PGQL 1.1	`SELECT a.name, l.name FROM languageGraph`
	`MATCH (a:Engineer)-[:LIKES]->(l:Language)`
	`WHERE EXISTS (`
	`  SELECT *`
	`  MATCH (l)-[:SUPPORTS]->(:Feature {name: "Pattern Matching"})`
	`)`

G-Core	`SELECT a.name, l.name`
	`MATCH (a:Engineer)-[:LIKES]->(l:Language) ON languageGraph`
	`WHERE EXISTS (`
	`  CONSTRUCT ()`
	`  MATCH (l)-[:SUPPORTS]->(:Feature {name: "Pattern Matching"})`
	`  ON languageGraph`
	`)`

Scope

DQL (projecting tables and grraph construction)

DML (updating base data and views)

DDL (schema, constraints, indices)

Path variables

Cypher (+**PGQL**?)	Immutable path values
G-Core	Mutable path objects

Path patterns

Full RPQs

Possibly: CFGs

Handling costs

Syntax variations ("inline" vs "nested")

Predicates in nested RPQs

Binding variables

Configurable matching semantics

Cardinality (existence/single, top-k, enumeration)

Path length restriction (shortest, cheapest, heuristic (?))

Morphisms

- Homomorphism
- Node-isomorphism
- Edge-isomorphism
- Explicit isomorphism

Graph construction and views

How to express shared entities?
reference semantics via global identities or via provenance tracking

How to express construction?
patterns (G-Core) vs DML (Cypher 10 proposals)

How to update views?
precise semantics and allowed operations

How to handle schema?
required / inferred / optional

Details

Convergence

- *Scope*
  All of it: DQL, DML, DDL :)

- *Pattern matching*
  mainly consensus except for extent of RPQ support and syntactic details

- *Path variables*
  accepted but concrete semantics needs to be decided

- *Configurable matching semantics*
  Shortest, cheapest, homomorphism, support isomorphism

- *Graph construction*
  accepted but details varying

Other open issues

- Order of clauses
  - Top-Down => Logical and proven for property graphs
  - Bottom-Up => Nice for SQL
- Default morphism
  - homomorphism by default for rigid patterns
  - shortest paths by default for variable length patterns
  - enumeration/reachability (?)
- Type system
  - "Nice subset" of SQL, GraphQL, SPARQL, Cypher, ...
  - Extensible
  - Unicode by default?
- Support for heterogenous typing and schema-optional graphs?

Other open issues

- Semantics of DML operations (e.g. `MERGE`)
- Schema model (Discussed as part of SQL work)
- SQL integration

Thank you

Link to paper