Towards GQL v1

A Property Graph Query Language Standard

15th LDBC Technical User Committee (TUC) Meeting
18 June, 2022

Petra Selmer, Query Language Standards & Research, Neo4j

petra.selmer@neo4j.com
Topics

- What is GQL?
- How is GQL produced?
- What does GQL look like?
Attention

● GQL is still under development and not final
  Features may be changed, dropped, or moved to a future version.

● ISO database standards are “featurized”
  Implementations are considered conforming as long as they don’t violate the standard but it’s up to them which optional features they choose to implement.

● Safe harbour statement
  Nothing in this talk, the slides, or the accompanying discussion represents a commitment by Neo4j (or any other vendor) to implement GQL or any of its features.
What is GQL?
What is GQL?

**Standardization effort** by "The SQL Committee" for a new graph query language.

Motivated by growing adoption of property graphs (fastest growing database segment by far) and commonalities across languages.

Initiated by A. Green's "The GQL-manifesto": open letter to database industry: "Let’s build a next generation, declarative, composable, compatible, modern, intuitive International Standard for a Property Graph Database Language" (Votes: 95 % of ca. 4000 votes: YES)
Property Graph Data Model

- Nodes (vertices) and relationships (edges) have
  - synthetic identity
  - 0..n labels
  - 0..n properties

- Edges are directed or undirected

- Graphs have
  - 0..n labels
  - 0..n properties
Visual Graph Pattern Syntax

MATCH (a:Person)-[:KNOWS*{1,2}]->(b:Person)
RETURN *

- Visual highly intuitive "Ascii-Art" syntax
- Use for property graph matching originally pioneered by Neo4j
- Idea adopted by openCypher, G-CORE, GSQL, PGQL
- "Best syntax for describing joins ever invented"
- Applicable in DQL, DML, DDL, Serialization

Figure 20: Query graph for “What was Lee’s age when hired?”

GQL Goals

1. **Industry effort** informed by research and by community requirements.

2. **Universal property graph query language** that users can depend on to access graph databases, enabling skills reuse, vendor interoperability, and data longevity.

3. Establish **graphs as primary data model**, raising the level of abstraction and thereby enabling graph views and transformation.

4. **Backwards compatible** with existing languages, applications, and skills. No idle variation from proven syntax & semantics.

5. **Query language for all**: graph experts, SQL users, programmers, and data analysts.

6. **Grow the property graph space** to enable use of connected data by modern organizations.

7. **Integrate into modern technology stacks**: Unicode, IEEE Floats, ISO 8601 Temporal data, ...

8. Standard that is **easy to learn, use, teach, implement, and evolve**.
Property Graph Standard GQL

GQL

- Full DB language
  - DQL - **Graph pattern matching** queries
  - DML – **CRUD** (Create, read, update, and delete) on graph elements and their labels and properties
  - DDL – Create graphs, **graph types**, etc. in a **global hierarchical database catalog**

- Optional extensions to the property graph model: multiple edge types, undirected edges
- Leverages common foundation from SQL and property graph languages, incl. **session and transaction control**
- Supports **schema-fixed** and **schema-optional** variants
How is GQL produced?
Property Graph Standards – SQL/PGQ and GQL

**SQL/PGQ**
- Property Graph views of SQL tables
- **Graph Pattern Matching queries**
  - GRAPH_TABLE() in SQL FROM
  - Supports Reads
- Common foundation with SQL and graph query languages
- Does not support schema-optional graphs

**GQL**
- Full DB language
  - DML – Create, Read, Update, Delete
  - DDL – Create Type, Create Graph
- **Graph Pattern Matching queries**
- Leverages common foundation from SQL and property graph languages
- Supports schema-fixed and schema-optional variants
Property Graph Standards – SQL/PGQ and GQL
GQL community work

GQL Community

LDBC

EL WG
PGS WG
FS WG

ISO/IEC JTC 1 Info Tech

SC32 Data Management and Interchange

WG3 Database Languages

1987 SQL
2019 GQL

gqlstandards.org

Towards GQL v1, Petra Selmer, LDBC TUC, 2022
Work on the GQL draft between ballots

Experts produce and agree to specific changes to the draft in form of change proposal documents

Experts debate and agree to change proposal documents

* Or experts in other national bodies or a liaison

Editors modify the draft accordingly

WG 3 Database Languages

Editors

Stefan Plantikow and Stephen Cannan
GQL Progress

- 505 pages with annexes, indexes, notes, released monthly
- Editorially drafted, currently reviewing/rewriting features
- Pattern matching functionality
- Execution model of the standard
- GQL-Environment and GQL-Catalog, data model, and basic graph schema
- Predefined data types
- Ongoing: Query structure, DML and DQL statements
- Ongoing: Type system
- Ongoing: Resolving issues and comments
- Ongoing: Reducing size and scope
GQL v1

- Go-to language for all new (and existing) property graph vendors
- We want to ensure adoption is as widespread as possible
- v1: focus on the core minimum
- Reduce feature set size => punt these to v2 and beyond

Start small => get big over multiple versions
What does GQL look like?
A taste of GQL: Multigraph query

CALL {
  FROM socNet.twitter
  MATCH (f:Follower)
  RETURN f, "twitter" AS kind
  UNION
  FROM socNet.instagram
  MATCH (f:Follower)
  RETURN f, "insta" AS kind
}
MATCH (c:Customers) WHERE c.email = f.email
RETURN c.name AS name, kind
(2) Pattern matching syntax extensions

- Selecting nodes and relationships with label expressions (and, or, not, etc.), e.g.: `Person & (Employee | Intern)`
- Path pattern union: `MATCH ( (a)-[:KNOWS]->(b) | (a)<-[[:LOVES]]-(b) )`
- Multiset alternation: `MATCH ( (a)-[:KNOWS]->(b) |+| (a)<-[[:LOVES]]-(b) )`
- Quantified path patterns
  - Simple: `MATCH (a:Boss)-->(b:Sales)`
  - Filtering: `MATCH ((a:Boss)-[r]->(b:Sales) WHERE r.age>5)`
  - Union: `MATCH ((a)-[:KNOWS]->(b) |+| (a)<-[[:LOVES]]-(b)){2, 6}`
  - Alternation: `MATCH ((a)-[:KNOWS]->(b) |+| (a)<-[[:LOVES]]-(b)){2, 6}`
  - Aggregation: `MATCH (a) ((:A)-[r:L]->(:C|D)){1,5} (b)`
    WHERE a.height < AVG(r.weight) AND AVG(r.weight) < b.height
  - + any combination

- Matching walks, trails, simple paths, top k shortest/paths/path groups, ...
Graph schema as a graph

// Graph type describing graph schema
(:Person { gender STRING, birthday DATE } ),
(:Message { creationDate DATETIME, context TEXT } ),
(:Tag { name STRING, url STRING } ),
...

(:Person)-[:LIKES { creationDate DATETIME }]->(:Message),
(:Message)-[:HAS_TAG]->(:Tag),
(:Person)-[:HAS_INTEREST]->(:Tag),
...

// Not yet defined
● Schema constraints
● Key constraints
● Cardinality constraints
...

Figure 2.1: The LDBC SNB data schema
CREATE GRAPH TYPE `SocialNetwork` {

(Person :Person {name STRING, dob DATE}),
(City :City:Town {name STRING}),

(Person)-[:LIVES_IN]->(City),
(Person)-[:KNOWS {year INT}]->(Person)
}

Labels
Schema
Node types
Property types
Relationship types

Type names: not necessary, but helpful for more complex schemas
A taste of GQL: DML

```gql
INSERT ()-[r:S]->()
SET r = { a: 20, b: "West", c: 0.937 }
RETURN r.a, r.b, r.c // 20, "West", 0.937

MATCH ()-[r { a: 20 }]->()
SET r.b = "West"
RETURN r.a, r.b // 20, "West"
```
A taste of GQL: SELECT

```
SELECT t.name AS team, avg(p.age) AS avgAge, count(p) AS numPlayers
FROM SportsGraph
MATCH (t:BasketballTeam)->(p:Player) WHERE t.level = 'pro'
GROUP BY t HAVING numPlayers > 5
ORDER BY avgAge DESC
LIMIT 5
```
GQL v1

Property Graph Query Language Standardization

- Standardization effort by "The SQL Committee" for a new graph query language.

```sql
FROM isoIecJtc1graph
MATCH (:ISO_WG {num: 3})-[:WORKS_ON]->(gql:Standard {num: 39075})
RETURN gql
```

- Standards are hard and take a while