GQL Scope and Features

(incl. GQL Process Update)

Stefan Plantikow for the Neo4j Query Languages Team, LDBC TUC Meeting, Amsterdam, Netherlands, 2019
Information technology — Database languages — GQL

- Next Generation Query Language for Property Graphs by ISO/IEC JTC 1/SC 32/WG 3 (Convenor: Keith Hare)
- Goal: ISO Standard
- Status: Ballot Proposal for New Work Item Proposal (NWIP) started (until: Sep 2019)

RESOLUTION 03-003 – New Work Item Ballot Circulation
SC 32 requests its Committee Manager to circulate the following proposal for a 12-week letter ballot and to JTC 1 for concurrent review.

<table>
<thead>
<tr>
<th>Designation</th>
<th>Title</th>
<th>Project Editor</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC 32 N3002</td>
<td>Information Technology – Database Language GQL</td>
<td>Stefan Plantikow,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stephen Cannan</td>
</tr>
</tbody>
</table>

Note: ISO/IEC 39075 has been reserved for this project.

"GQL Scope and Features (incl. GQL Update)". S. Plantikow for Neo4j Query Languages @ LDBC TUC Meeting Amsterdam, 2019
Motivation

① Lead and consolidate existing demand for such a language
② Address the specific needs of graph use cases
③ Increase the utility of property graph querying
④ Drive adoption of graph database systems
GQL Early Working Draft

- Currently being written
  - Sharing foundational aspects with SQL
  - Using same conventions as SQL and tooling developed by Jim Melton and other SQL editors
- Available to standards process participants only at this time
- Starting point that will see many changes
- Based on
  - GQL Scope and Features (*this talk*)
  - Other WG3 contributions
From Cypher, PGQL, GSQL, SQL/PQ to **GQL**

- **Simple Pattern Matching**
- **Complex Pattern Matching**
  - (RPQs, Shortest/Cheapest Path, Macros)
- **Tables out only**
- **Graphs, tables, scalars in/out**
- **Single graph only**
- **Multiple graphs & (parameterized) views**
- **DML only**
- **DML, Graph computation, Graph projection**
- **No schema**
- **Schema & advanced type system**

All aligned with basic data types, infrastructure, and expressions of the SQL database
Support for basic tabular manipulation (projection, sorting, grouping etc)
More features are discussed (Indexing)

BNE-023: GQL Scope and Features

1. Introduction
   Inputs, motivation, orientation

2. References
   Related material, incl. designs from openCypher

3. Discussion
   Overarching design principles, language overview

4. Proposal
   Project scope, definitions, language features

5. Grammar
   Sketch of proposed syntax - to show structure

Publically available at http://tiny.cc/gql-scope-and-features
BNE-023: GQL Design Principles and Scope [3.9, 4.1]

① New and independent property graph query language
   Follow the tradition of existing languages like Cypher, PGQL and SQL/PGQ
   Support both standalone implementation or extension of existing SQL-based systems

② Declarative language
   Emphasize what over how (in particular by using pattern matching) to support different implementation strategies

③ Composable language
   Compose procedures from nested sub-procedures and statement sequences in a language closed under graphs and tables

④ Compatible language
   Ensure compatibility with established and widely used features of SQL and avoid idle variation to existing syntax

⑤ Modern language
   Introduce next-generation language features using established designs from existing languages where available

⑥ Intuitive language
   Follow consistent, "whiteboard-friendly", visual syntax (in particular by using "ascii-art" patterns)

Covering the full spectrum of features of an industry-grade database query language!
BNE-023: Example query [3.1]

```
// from graph or view "friends" in the catalog
FROM friends

// match persons a and b that travelled together
MATCH (a IS Person)-[IS TRAVELLED_TOGETHER]-(b IS Person)
WHERE a.age = b.age AND a.country = $country AND b.country = $country

// from view parameterized by country
FROM census($country)

// find out if a and b at some point moved to or where born in a place p
MATCH SHORTEST (a) (()-[IS BORN_IN|MOVED_TO]->())* (p)
  (())<-[IS BORN_IN|MOVED_TO]-()*(b)

// that is located in a city c
MATCH (p)-[IS LOCATED_IN]->(c IS City)

// aggregate number of such pairs per city and age cohort
RETURN a.age AS age, c.name AS city, count(*) AS pairs GROUP BY age
```

"GQL Scope and Features (incl. GQL Update)". S. Plantikow for Neo4j Query Languages @ LDBC TUC Meeting Amsterdam, 2019
BNE-023: Topics

1. Graphs & Pattern matching
2. Tables & Expressions
3. Type system & Schema
4. Modifying and Projecting graphs
5. Query composition & Views
6. Schema & Catalog
7. Interoperability
8. Error handling model
9. Security model
10. User defined procedures & functions
BNE-023: Graphs & Pattern matching [3.4, 4.6]

1. Property graph model
2. Nodes, Edges, and Paths [4.4.3]
3. Patterns [ERF-035, BNE-034]
5. Working with paths [4.6.5]
BNE-023: Intrinsic Identity
MATCH (query)-[:MODELED_AS]->(drawing),
  (code)-[:IMPLEMENTS]->(query),
  (drawing)-[:TRANSLATED_TO]->(ascii_art),
  (ascii_art)-[:IN_COMMENT_OF]->(code),
  (drawing)-[:DRAWN_ON]->(whiteboard)
WHERE query.id = {query_id}
RETURN code.source
**BNE-023: Pattern matching modifiers**

<path modifiers> for controlling path matching semantics

[ALL] **SHORTEST** - for shortest path patterns
[ALL] **CHEAPEST** - for cheapest path patterns
(both with **TOP** <k>, **MAX** <k> qualifiers, and supporting **WITH TIES**)

**REACHES** - unique end nodes with >=1 matching path
**ALL** - all paths

**SIMPLE** - may not contain repeated nodes
**TRAIL** - may not contain repeated edges
**ACYCLIC** - may not repeat nodes, except allowing the first and last node to be the same

FROM twitter
MATCH SIMPLE (a)((IS Knows)->())* (b), TRAIL (a)-[IS Lives_At]->()
(()-[IS Bus|Train|Plane]->())* ()<-[IS Lives_At]-(b)
BNE-023: Pattern matching structure

[FROM <graph>]
MATCH <pattern> {<comma> <pattern> ...}

+ optional modifiers to MATCH
for controlling pattern matching behaviour

OPTIONAL MATCH - outer join, binds nulls if nothing matches
MANDATORY MATCH - query fails if nothing matches

MATCH ...[
● DIFFERENT (VERTICES|NODES) - vertex isomorphism
● DIFFERENT (EDGES|RELATIONSHIPS) - edge isomorphism
● UNCONSTRAINED - homomorphism
]

FROM twitter
MATCH (a)-[IS Follows]->(b)

OPTIONAL MATCH ( 
  (b)-[p IS Posted]->(m) 
  WHERE p.date > three_days_ago 
)
BNE-023: Tables & Expressions [4.5, 4.9]

1. Basic table operations (selection, projection, ordering, filtering, slicing) [4.9]
2. Aggregation and grouping [4.5]
3. Tabular set operations (\texttt{UNION [ALL ]}) [4.9]
4. Graph element expressions [4.5]
5. Collection and dictionary expressions [4.5]
6. Relationship to SQL
BNE-023: Why tabular operations in GQL?

(A) Pattern matching => (Multi) set of bindings (=> Table) => *Tabular result transformation useful to avoid client-side processing*

(B) Bindings main input into graph modifying operations (DML) => *Supported by tabular result transformation and combination*

(C) Bindings main input into graph construction operators => *Supported by tabular result transformation and combination*

**Not needed:** Features focussed on tables as a base data model like e.g. referential integrity via foreign key constraints
BNE-023: Linear statement composition [3.10.3, 4.3.4.3]

- Top-Down flow
- Combined using lateral join
- Statements are update horizons

Benefits
- Natural, linear order used in programming
- Allows query-aggregate-query without (named) nested subqueries
- Allows mixing reading and writing (e.g. returning modified data)
- Solvable using subquery unnesting (maps on "apply" operator)
- **RETURN** has been very positively received by PGM users

"GQL Scope and Features (incl. GQL Update)", S. Plantikow for Neo4j Query Languages @ LDBC TUC Meeting Amsterdam, 2019
BNE-023: Graph element expressions and functions

- Element access: \texttt{n.prop}, \texttt{labels(n)}, \texttt{properties(n)}, \texttt{handle(n)}
- Dynamic label tests
- Element operators: \texttt{allDifferent(<elts>)}, =, \texttt{<>}
- Element functions: \texttt{source(e)}, \texttt{target(e)}, \texttt{(in|out)degree(v)}
- Path functions: \texttt{nodes(p)}, \texttt{edges(p)}, concatenation
BNE-023: Collection and dictionary expressions

- Collection literals: `[ a, b, c, ... ]`
- Dictionary literals: `{ alpha: some(a), beta: b+c, ... }`
- Indexing and lookup: `coll[1], dict[‘alpha’]`
- Map comprehensions
- List comprehension
- Functions
BNE-023: Type system & Schema [3.3, 4.4]

1. Selected scalar data types from SQL [4.4.1]
2. Nested data and collections [4.4.2]
3. Graph-related data types [4.4.3]
   - Nodes and Edges - with intrinsic identity
   - Paths
   - Graphs
4. Advanced type system features [3.3, 4.4.4]
5. Static and dynamic typing [4.4.5]
BNE-023: Advanced types

Heterogeneous types
MATCH (n) RETURN n.status may give conflicting types (esp. in a large schema)
Possible type system extension: Union types, e.g. A | B | NULL

Complex object types
Support the typing of complex objects like graphs and documents
Possible type system extension: Graph types, structural types, recursive document type

Incomplete type information
Data access (e.g. in a big data file system) with runtime metadata discovery
Possible type system extension: Gradual type for "value of unknown type"?
BNE-023: Modifying and Projecting graphs

① Modifying graphs using patterns [3.5, 4.7]
② Graph projection [4.8]
③ Element sharing [4.8]
④ Graph combinators (UNION, INTERSECTION, ...) [4.8]
BNE-023: Graph projection

- Deriving identical elements in the projected graph ("sharing")
- Deriving new elements in the projected graph
- Shared edges always point to the same (shared) endpoints in the projected graph
BNE-023: Graph Projection is inverse pattern matching
BNE-023: Query composition & Views

① Composable graph procedures [4.3.3-4.3.5]  
② Parameters and results [4.3.2, 4.3.4.1]  
③ Linear statement composition [3.10.3, 4.3.4.3]  
④ Graph views model [3.7]  
⑤ Updatable views and graph augmentation  
⑥ Provenance tracking
BNE-023: Queries are procedures [4.3]

- Use the output of one query as input to another to enable abstraction and views
- Both for queries with tabular output and graph output
- Nested queries and procedures [4.10]
- Simple linear composition of tabular output of one query as input to another [3.10.3]
BNE-023: Views [3.7, 4.12]

- Graph elements in views are derived from other graphs (which may again be views)
- Graph elements are "owned" by their base graph or introducing views
- Derivation graph must form a DAG
- Updates reverse transformation
BNE-023: Views [3.7, 4.12]

- A (graph) view is a query† that returns a graph
  - GQL could also support tabular views
- A view can be used as if it was a graph
  - A tabular view can be used as if it was a table
- Queries (incl. views) can be parameterized
  - allowing the application of the same transformation over compatible graphs

CREATE QUERY foaf($input SocialGraph) AS {
  FROM $input
  MATCH (a)-[IS FRIEND]-(b)-[IS FRIEND]-(b)
  CONSTRUCT (a)-[IS FOAF]-(b)
}

FROM foaf(facebook) MATCH ...
FROM foaf/twitter) MATCH ...
Views behave as if conceptually computed on the fly, including shared graph elements but what if one wants to explicitly express persistently shared graph elements?

Graph augmentation: Allow explicit persistent layered graphs with derived graph elements

Many open questions (e.g. deletion semantics, security model implications)
BNE-023: GQL Scope and Features

A new and independent

Declarative,

Composable,

Compatible,

Modern,

Intuitive

Property Graph Query Language

http://tiny.cc/gql-scope-and-features

http://tiny.cc/gql-scope-digest