GQL V1 Overview

Fundamentals, Features, Future

Stefan Plantikow,
GQL Editor, ISO/IEC JTC1/SC32/WG3 Database languages

stefan.plantikow@neo4j.com

16th LDBC TUC Meeting
23 June 2023
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?

- Information technology — Database languages — **GQL**
  ISO/IEC JTC1/SC32/WG3 39075
- A **new graph query language standard** by the "ISO/IEC SQL-Committee"; now at DIS (draft) stage: GQL is technically mostly complete!
- Initiated by A. Green's "**The GQL-manifesto**" and motivated by growing property graph adoption and graph query language commonalities.

A complete database language: **DQL, DML, DDL**

**Syntax**  
SELECT-style and RETURN-style following SQL, Cypher, PGQL, GSQL, and G-Core

**Execution model**  
sessions, transactions, and requests

**Data model**  
labeled property graphs in a hierarchical catalog

**Access paradigm**  
pattern matching into binding tables

**Schema model**  
mandatory schema and schema-free

**Data types**  
based on SQL, Unicode, IEEE 754, and ISO 8601
Complete database language

GQL DQL
(query procedures)
AT <schema>
USE <graph>
MATCH <pattern>
LET <var> = <expr>
FOR <var> in <list-expr>
FILTER <predicate>
ORDER BY .. OFFSET|LIMIT <n>
RETURN|SELECT ... [ GROUP BY .. ]

GQL DML
(data procedures)
INSERT (:Person { name: "Jane" })
SET n:Label
REMOVE n:Label
SET n.prop = 42
REMOVE n.prop
[ NODETACH|DETACH ] DELETE n

GQL DDL
(catalog procedures)
CREATE|DROP
  SCHEMA
  |GRAPH
  |GRAPH TYPE
  |...

GQL SESSION COMMANDS
SESSION SET
SESSION RESET
SESSION CLOSE

GQL TRANSACTION COMMANDS
START TRANSACTION
ROLLBACK
COMMIT

GQL COMPOSITION
CALL <subquery>
CALL <procedure>
NEXT

A taste of GQL (1)

```
SESSION SET $country = 'MA'  /* session parameters */
START TRANSACTION  /* transaction demarcation */

① USE socialGraph    /* which graph to query */
② MATCH (p:Person)-[:FRIEND]->()-[:FRIEND]->(f:friend)  /* match a pattern */
    WHERE p.age < f.age AND f.country = $country    /* with a filter */
③ INSERT (p)-[:FOAF]->(f)    /* INSERT new data */
④ RETURN count(*) AS edges_added    /* Supports SELECT, too */

COMMIT  /* transaction demarcation */
SESSION CLOSE  /* session demarcation */
```
A taste of GQL (2)

```sql
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: A standard for many implementations

- Support **different implementations**.
- High degree of **featurization**.
- **Minimal implementation:**
  - Single (ambient) graph.
  - Minimal set of essential data types.
  - Basic (join-like) pattern matching.
  - Read-only transactions.
- Expose any data as a property graph!
Execution model

1. **GQL-agent** instructs **GQL-client** to send a **GQL-request** to the **GQL-server** on behalf of the **User**

2. **GQL-server** executes the **GQL-request** in the current session

3. **GQL-server** delivers execution outcome to **GQL-agent** via the **GQL-client**

**GQL-request**
- GQL-request source (valid GQL-program)
- GQL-request parameters (unique name-value pairs)

**GQL-program**
- session commands, transaction commands, and procedures (catalog procedures, data procedures, and query procedures) composed from statements serially execute in the current session and transaction
Execution of commands and statements

1. Execution context provides:
   - Working record (fixed variables)
   - Working table (iterated variables)
   - Execution outcome (status + opt. result)

2. Status contains:
   - GQLSTATUS code
   - Optional diagnostic information
   - Nested causes

3. Possible results currently are:
   - Binding tables
   - Values (incl. reference values)
   - Omitted (On successful DML)
Diagnostics and status codes

RECORD {
  COMMAND_FUNCTION: CF,
  COMMAND_FUNCTION_CODE: CFC,
  CURRENT_SCHEMA: CS,
}

<table>
<thead>
<tr>
<th>Category</th>
<th>Condition</th>
<th>Class</th>
<th>Subcondition</th>
<th>Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>graph does not exist</td>
<td>G03</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>graph type does not exist</td>
<td>G04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>null value eliminated in set function</td>
<td>G11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>no data</td>
<td>02</td>
<td>(no subclass)</td>
<td>000</td>
</tr>
<tr>
<td>X</td>
<td>connection exception</td>
<td>008</td>
<td>(no subclass)</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>transaction resolution unknown</td>
<td>007</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>data exception</td>
<td>22</td>
<td>(no subclass)</td>
<td>000</td>
</tr>
<tr>
<td></td>
<td>string data, right truncation</td>
<td>001</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>numeric value out of range</td>
<td>003</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>null value not allowed</td>
<td>004</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>invalid datetime format</td>
<td>007</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>datetime field overflow</td>
<td>008</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>substring error</td>
<td>011</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>division by zero</td>
<td>012</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>interval field overflow</td>
<td>015</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>invalid character value for cast</td>
<td>018</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>invalid argument for natural logarithm</td>
<td>01E</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>invalid argument for power function</td>
<td>01F</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>trim error</td>
<td>027</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hierarchical GQL-catalog & GQL-data heap

/Directory
  /Subdirectory 1
    ...
  /Subdirectory n
  /Schema 1
    MyGraph ===> Graph object
    MyGraphType ===> Graph type object
    MyProcedure ===> Procedure object
    ...

  /Schema 2
    ...

- Exact structure of GQL-catalog left to implementations.
- Conceptual separation between catalog entries and data objects.
Data model: labeled property graphs

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

- Edges (relationships) are either **directed** or **undirected**

- Model instances may be restricted by
  - **constraining graph type**
  - implementation limits on number of labels, number of properties, and supported property value types
Access paradigm: Pattern matching (1)

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

- Visual highly intuitive "Ascii-Art" syntax
- "Best syntax for describing joins ever invented"
- Use for property graph matching originally pioneered by Neo4j
- Idea adopted by openCypher, G-CORE, GSQL, PGQL
- Applicable in DQL, DML, DDL, Serialization
Access paradigm: Pattern matching (2)

- Shared between GQL and SQL/PGQ
- Core features:
  - **natural join**, e.g.
    
    (a)->(b), (a)->(x)
  - **label expressions**, e.g.
    
    :Person&(Employee|Intern)
  - **filtering** with predicates and restrictors, e.g.
    
    TRAIL (a)-[:FRIEND]->(b)-[:FRIEND]->(c) WHERE a.born > b.born AND c.born > b.born
  - **bounded length**, e.g.
    
    () -[]->*{1,2} ()
Access paradigm: Pattern matching (3)

- Multiple semantics: **all paths**, **shortest paths**, **different edges** (aka edge-isomorphism)
- **Unbounded transitive closure**, e.g. () -[]->[*] ()
- **Nested pattern matching** with optional filtering and aggregation on **group variables**, e.g.
  
  \[(a)(-[:X]->(r)[-[:Y]-> WHERE r.score > 0.5]*(b) WHERE sum(r.score) > 50)\]

- **Path pattern union**, e.g.

  \[(a) ( -[:KNOWS]-
  \ | -[:WROTE]->()<-[:WROTE]-
  \ | -[:WORKS_AT]->()<-[:WORKS_AT]- ) (b)\]

- **Path binding**, e.g.

  \[p=()->()\]
Binding tables

- Main container of intermediate tabular results.
- Drive iteration and linear ("flat map and filter") composition of most statements.
- Collection of records of the same record type:
  - No duplicate columns
  - No positional columns
  - Associated column order is tracked purely as metadata for client-side use
- Either ordered or unordered:
  - Order needs to be established explicitly
  - Order only preserved until next statement
GQL type system

- **Provides:**
  - Static typing using graphs with mandatory schema and
  - Dynamic typing using schema-free graphs

- **Approach:**
  - Choice between static typing, dynamic typing, or both
  - Optional constraining types for data objects
  - Open (unrestricted) vs closed (specific) value types
  - Built on SQL-compatible foundations
Graph types

Optionally restrict the contents of graphs

CREATE GRAPH messaging ::

GRAPH {
   (: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),
   ...
}

Value types

- **Foundations**: Compatible subset of predefined types from SQL
  - Unicode character strings
  - Byte strings
  - Numbers (base 2 integers, base 10 decimals, IEEE 754 aligned floats)
  - Booleans

- **Native nested data**: Records (structs) and lists

- **Object references**: Graphs, paths, nodes, edges, binding tables, ...
Multigraph workflows

GQL support complex graph processing workflows across multiple graphs via

- Procedure composition
  (named procedures, subqueries)
- Inter-statement composition
  (linear binding table composition)
- Intra-statement composition
  (expressions, predicates)

```
USE customers
CALL {
    USE /socNet/twitter
    MATCH (f:Follower)
    RETURN f, "twitter" AS kind
    UNION
    USE /socNet/instagram
    MATCH (f:Follower)
    RETURN f, "insta" AS kind
}
MATCH (c:Customers)
WHERE c.email = f.email
RETURN c.name AS name, kind
```
Future

- Getting GQL out: Implementations! Implementations! Implementations! (and implementation adoption!)
- Many things left to do (e.g., see DCA-031/LEX-036, other LDBC DCA papers)
- **Personal** feature shortlist based on completeness/urgency:
  - Schema-related extensions: Alteration, constraints, keys, computed and default properties, related expressions, ...
  - Nested data support: Both natively and via JSON, comprehensions, path expressions, schema-level verification, and related types (e.g., UUID type)
  - Support for analytics and AI: Graph projection/views, model management, UDPs, ...
Summary

- **GQL**: A new graph database language standard
- **Status**: Draft International Standard
- **554 pages with annexes and indexes** (fully reworked initial 350 page editors' draft)
- **GQL digital artifacts** (grammar, status and error codes, feature codes...) are freely available from ISO
- **Aim to finish end of 2023**

Big thank you to everyone helping making this happen (ISO WG3 and NBs, vendors, LDBC community, ...)!