Graph Databases

Guillaume Raschia — Nantes Université

Last update: October 19, 2023

Contents

Graph DB Landscape

“Old School” Graph Database Style

Model

A graph $G(V, E)$ is a binary relation $R(\text{src}, \text{dst})$

src	dst
1	2
1	3
3	1

Queries

- Relational Algebra (**procedural** language): $\{\sigma, \pi, \bowtie, \rho, \cup, -\}$
- 3-hops: $R \underset{\text{dst1}=\text{src2}}{\bowtie} R \underset{\text{dst2}=\text{src3}}{\bowtie} R$

Limitations

- Join $\bowtie$ is the key operator: costly!
- Reachability: $\bigcup_k R \bowtie_1 R \bowtie_2 \ldots \bowtie_k R$: **Recursion** and **fixpoint**


neo4j
The #1 Database for Connected Data

Graph DB (cont'd)

Graph Data Models

Requirements for Graph Databases

The 3D graph data model [Angles et al., ACM CS 2008]

1. Data structure
   - data and schema as (distinct) **graphs**
   - standard abstractions: is-a, is-part-of, is-associated-to
2. Update and query language
   - graph transformations
   - primitives on paths, neighborhoods, subgraphs, graph patterns, connectivity and graph statistics (diameter, centrality, etc.)
   - multi-relational graph algorithms
3. Integrity constraints
   - schema-instance consistency, identity, referential integrity

Requirements for Graph Databases (cont'd)

Definition (Graph database (tentative of))

Any storage system that provides index-free adjacency

- Each vertex has direct references to its adjacent vertices
  - act as a **mini-index**
- $\mathcal{O}(1)$ to move from a vertex to its neighbors
- $\mathcal{O}(\log n)$ b.t.w. of an index in non-graph db's

Very large graphs such like TAO Social Graph at Facebook: 5 Billions+ nodes!

Bronson et al. (2013). *TAO: Facebook's Distributed Data Store for the Social Graph*. USENIX ATC.

Audrey Cheng et al. (2021). *RAMP-TAO: Layering Atomic Transactions on Facebook's Online TAO Data Store*. PVLDB 14(12): 3014-3027.

Abstract: Facebook's graph store TAO, like many other distributed data stores, traditionally prioritizes availability, efficiency, and scalability over strong consistency or isolation guarantees to serve its large, read-dominant workloads. As product developers build diverse applications on top of this system, they increasingly seek transactional semantics. However, providing advanced features for select applications while preserving the system's overall reliability and performance is a continual challenge. In this paper, we first characterize developer desires for transactions that have emerged over the years and describe the current failure-atomic (i.e., write) transactions offered by TAO. We then explore how to introduce an intuitive read transaction API. We highlight the need for atomic visibility guarantees in this API with a measurement study on potential anomalies that occur without stronger isolation for reads. Our analysis shows that 1 in 1,500 batched reads reflects partial transactional updates, which complicate the developer experience and lead to unexpected results. In response to our findings, we present the RAMP-TAO protocol, a variation based on the Read Atomic Multi-Partition (RAMP) protocols that can be feasibly deployed in production with minimal overhead while ensuring atomic visibility for a read-optimized workload at scale.

Graph Partitioning

Usually for a particular (set of) operation(s)

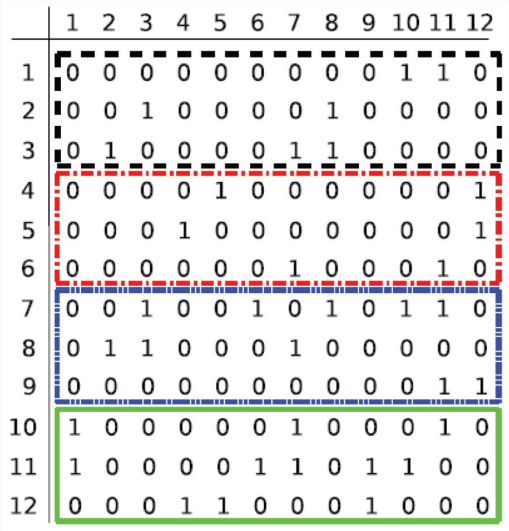
- The shortest path, finding frequent patterns, BFS, spanning tree search, ...

Graph partitioning is counter-intuitive and graph DB are mainly centralized

1-Dimensional Partitioning

- Partitioning of the adjacency matrix
- Focus on Breadth-First Search

1D Partitioning



	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0	0	0	0	0	0	0	0	1	1	0
2	0	0	1	0	0	0	0	1	0	0	0	0
3	0	1	0	0	0	0	1	1	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	1
5	0	0	0	1	0	0	0	0	0	0	0	1
6	0	0	0	0	0	0	1	0	0	0	1	0
7	0	0	1	0	0	1	0	1	0	1	1	0
8	0	1	1	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1
10	1	0	0	0	0	0	1	0	0	0	1	0
11	1	0	0	0	0	1	1	0	1	1	0	0
12	0	0	0	1	1	0	0	0	1	0	0	0

- Matrix rows are randomly assigned to the nodes (processors)
- Each vertex and the edges emanating from it are owned by one processor

1D Partitioning (cont'd)

BFS with 1D partitioning

1. Each processor has a set of frontier vertices F
2. The neighbors of the vertices in F form a set of neighbouring vertices N
    - Some owned by the current processor, some by others
3. Messages are sent to all "neighbouring processors", etc.

- 1D partitioning yields to high messaging
- Then, 2D-partitioning of adjacency matrix
    - lower messaging but still very demanding...

Efficient sharding of a graph is a hard problem

Dominant and Alternative Models of Graph DB's

Data model: Labeled Property Graph

A (attributed) multi-relational labeled digraph $G(V, E)$ where

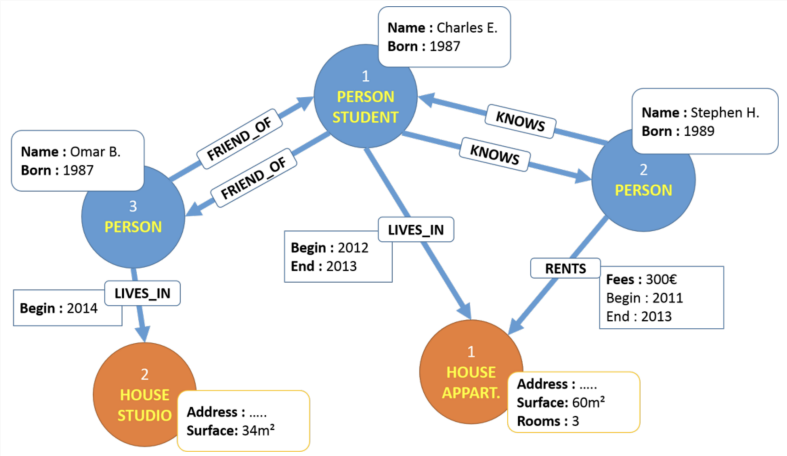
- V is a finite set of nodes (id's)
- $E \subseteq V \times \Sigma \times V$ are directed edges –**relationship**
- Σ is a finite alphabet of **labels** assigned to both nodes and edges
- Optional attributes –**properties**– may apply to both vertices and edges as a set of key/value pairs

Extensions to

- hypernode: nested graphs
- hypergraph: with hyperedges, i.e., sets of nodes
- multigraph: multiple single-relational edges

Labeled Property Graph Example

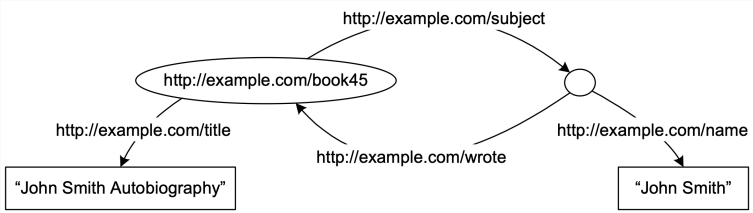
Node labels are “groups” whereas edge labels denote compulsory relationships



Source: R. Bouhali & A. Laurent (2015). *Exploiting RDF Open Data Using NoSQL Graph Databases.* AIAI pp.177-190.

The Web of Data

Resource Description Framework (RDF) Graph, aka. **Knowledge Graph**



Source: C. Sayers & A.H. Karp (2004). Computing the digest of an RDF graph.

- Data Model for the Semantic Web
- RDF statement: `(subject, predicate, object)`
- **Triple store** is a “flavor” of graph db

The Web of Data (cont'd)

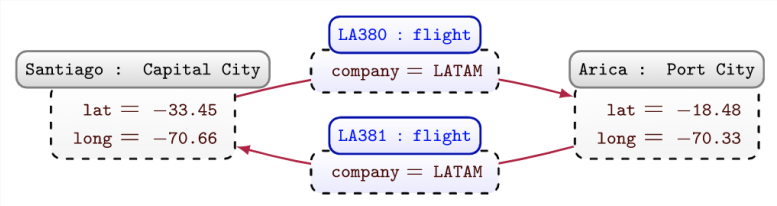
RDF Data Model: Edge-Labeled Digraph

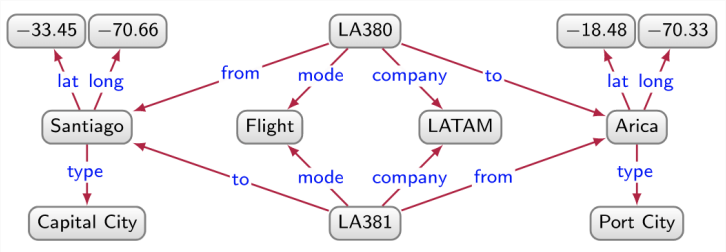
Vertex set V is split into URIs (U), literals (L), and blank/anonymous nodes (B), such that:

$$E \subseteq (U \cup B) \times U \times (U \cup B \cup L)$$

Extension to **named graphs** as ng $= (n, g)$ with $n \in U$ and $g \in \mathcal{G}$, the family of RDF graphs

Labeled Property Graph vs. Edge-Labeled Digraph

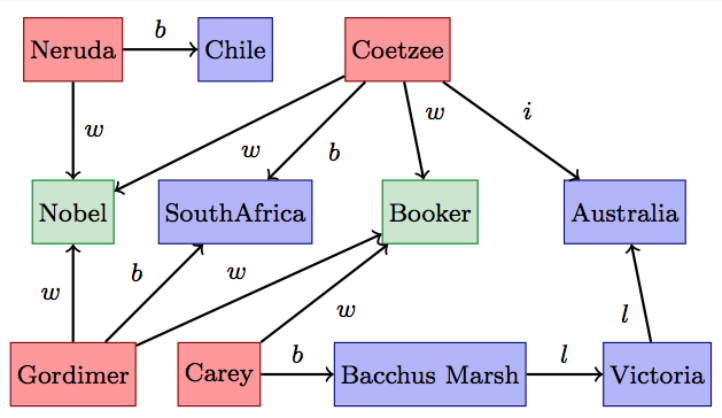




Source: A. Hogan et al. (2021). *Knowledge Graphs*, Chapter 2.

Graph Queries

Another Graph DB Example



b: bornIn w: hasWon i: livesIn l: locatedIn

Source: P.T. Wood, SIGMOD Record 2012

Conjunctive Queries

Implements **subgraph matching**

Definition (Graph CQ's)

$$Q(\vec{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, a_i, y_i)$$

where x_i, y_i are node variables or constants, each z_j is x_i, y_i or any constant, and $a_i \in \Sigma$

Example

$$Q(x) \leftarrow (x, \text{hasWon}, \text{Nobel}), (x, \text{hasWon}, \text{Booker}), (x, \text{bornIn}, \text{SouthAfrica})$$

Conjunctive Regular Path Queries

Implements **reachability** by path expressions

Definition (Graph CRPQ's)

$$Q(\vec{z}) \leftarrow \bigwedge_{1 \leq i \leq m} (x_i, r_i, y_i)$$

Extend CQ's to r_i as a **regular expression** over Σ

Example

$$\begin{aligned} Q(x) &\leftarrow (x, \text{hasWon}, \text{Booker}), (x, r, \text{Australia}) \\ r &:= \text{citizenOf} \mid ((\text{bornIn} \mid \text{livesIn}).\text{locatedIn}^*) \end{aligned}$$

Extended CRPQ's

Paths may

- occur in the output by means of **free path variables**
- be compared within a **regular relation**

Examples

Retrieve every path between nodes r and s that go through node e :

$$Q(\pi_1, \pi_2) \leftarrow (r, \pi_1, e), (e, \pi_2, s)$$

Retrieve all pairs (x, y) connected by paths following pattern $a^n.b^n$:

$$Q(x, y) \leftarrow (x, \pi_1, z), (z, \pi_2, y), a^*(\pi_1), b^*(\pi_2), (\bigcup_{a,b \in \Sigma} (a, b))^*(\pi_1, \pi_2)$$

Extended CRPQ's (cont'd)

Definition (Graph ECRPQ's)

$$Q(\vec{z}, \vec{\chi}) \leftarrow \bigwedge_{1 \le i \le m} (x_i, \pi_i, y_i), \bigwedge_{1 \le j \le p} R_j^{k_j}(\vec{\omega_j})$$

where χ_ℓ, π_i, ω_{jt} are path variables, and $R_j^{k_j}$ is a regular expression that defines a regular relation over Σ

Extension to approximate matching and ranking

- Define an **edit distance** $d_e(x, y)$ on Σ^*
- Operate with a regular expression over triples of the form (a, k, b), $a, b \in \Sigma \cup \{\epsilon\}$, k the cost of substitution

Aggregation and Arithmetic Predicates

count sum min max $+$ $-$ $*$ / for computing:

- degree, eccentricity of a node
- distance between two nodes
- diameter of the graph
- *etc.*

Example

Length of the *shortest path* between each pair of nodes

$$
\begin{array}{lcl}
\mathsf{len}(x, x, x, 0) & \leftarrow & \mathsf{dist}(x, y, \ell) \\
\mathsf{len}(x, x, x, 0) & \leftarrow & \mathsf{dist}(y, x, \ell) \\
\mathsf{len}(x, z, y, d) & \leftarrow & \mathsf{sp}(x, z, s), \mathsf{dist}(z, y, \ell), d = s + \ell \\
\mathsf{sp}(x, y, \min(d)) & \leftarrow & \mathsf{len}(x, z, y, d)
\end{array}
$$

SPARQL

Graph pattern-based SQL-like query language for RDF triple stores

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

The Jungle of Languages for Property Graphs

No(t yet any) Standard

Alternatives

- Native API in generic programming language

  depends on PL and Graph DB

- Procedural—algebraic-based—language

  depends on Graph DB

- Declarative—logic-based—language

  depends on Graph DB

Neo4J Native Java API

```
Node person = matching.getSingle();
Iterable<Relationship> relations =
    startNode.getRelationships(Direction.OUTGOING, RelTypes.FRIEND);
for (Relationship rel : relations)
{
    Node friend = rel.getEndNode();
    String name = friend.getProperty("name");
}
```

Source et planches suivantes: F. Holzschuher and R. Peinl, EDBT/ICDT 2013

Gremlin


```
t = new Table();
x = [];
g.idx("persons")[[id:id_param]].out("FRIEND_OF").fill(x);
g.idx("persons")[[id:id_param]].out("FRIEND_OF").
    out("FRIEND_OF").dedup().except(x).id.as("ID").
    back(1).displayName.as("name").
    table(t,["ID","name"]){it}{it}.iterate();
```

Cypher Query Language (CQL)

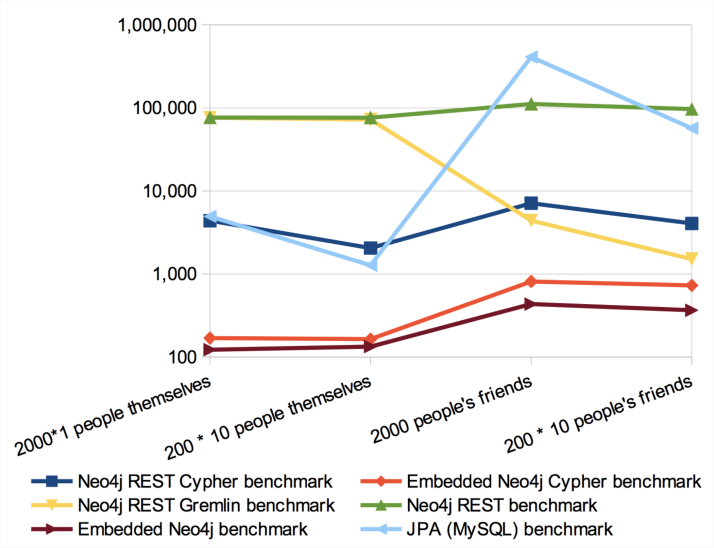
```
START person=node:people(id = {id})
MATCH person-[:FRIEND_OF]->friend-[:FRIEND_OF]->friend_of_friend
WHERE not (friend_of_friend<-[:FRIEND_OF]-person)
RETURN friend_of_friend, COUNT(*)
ORDER BY COUNT(*) DESC
```

Neo Technology released CQL to open source: see OpenCypher Project

SQL

```
SELECT persondb0.ID, persondb0.display_name
FROM person persondb0
WHERE persondb0.oid IN (
    SELECT frienddb2.friend_id
    FROM person persondb1, friend frienddb2
    WHERE persondb1.oid=frienddb2.person_id AND
    (persondb1.person_id IN (?))
    ) ;
```

Performance Comparison



Cypher Use Case

User story

As an employee,

I want to know who in the company has similar skills to me

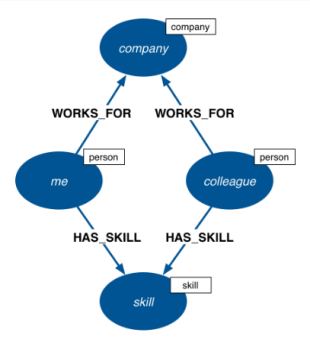
So that we can exchange knowledge

Query

Which people, who work for the same company as me, have similar skills to me?

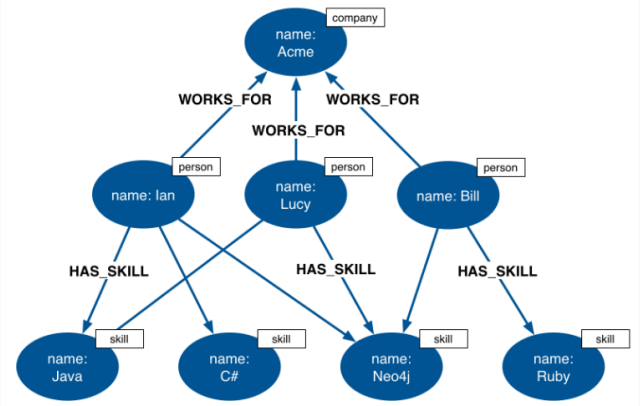
Query as a Graph Pattern

```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
    (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name= {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```



Running the Query

```
+-----------------------------------+
| name   | score | skills           |
+-----------------------------------+
| "Lucy" | 2     | ["Java","Neo4j"] |
| "Bill" | 1     | ["Neo4j"]        |
+-----------------------------------+
2 rows
```



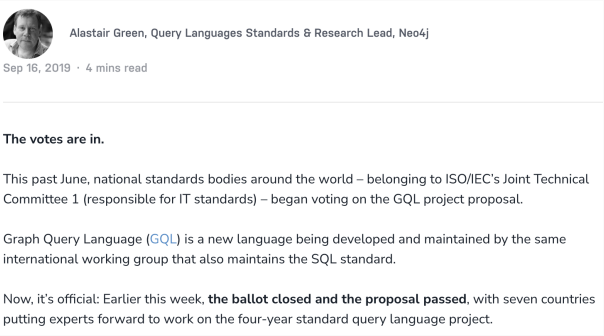
Examples

```
()
(x)--(y)
(m:MOVIE)-->(a:ACTOR)
(:MOVIE)-->(a { name: "Ivan Trojan" })
()<-[r:PLAY]-()
(m)-[:PLAY { role: "Ivan" }]->()
(:ACTOR { name: "Ivan Trojan" })-[:KNOW *2]->(:ACTOR)
()-[:KNOW *5..]->(f)
```

Towards a Standard Declarative Language for Graph DB's

Graph Query Language (GQL) is a Global ISO/IEC Standards Project alongside SQL[1]

- `gqlstandards.org` Last Update: October 4th, 2023
- A. Deutsch et al. (2022). *Graph Pattern Matching in GQL and SQL/PGQ*. SIGMOD



Alastair Green, Query Languages Standards & Research Lead, Neo4j

Sep 16, 2019 · 4 mins read

The votes are in.

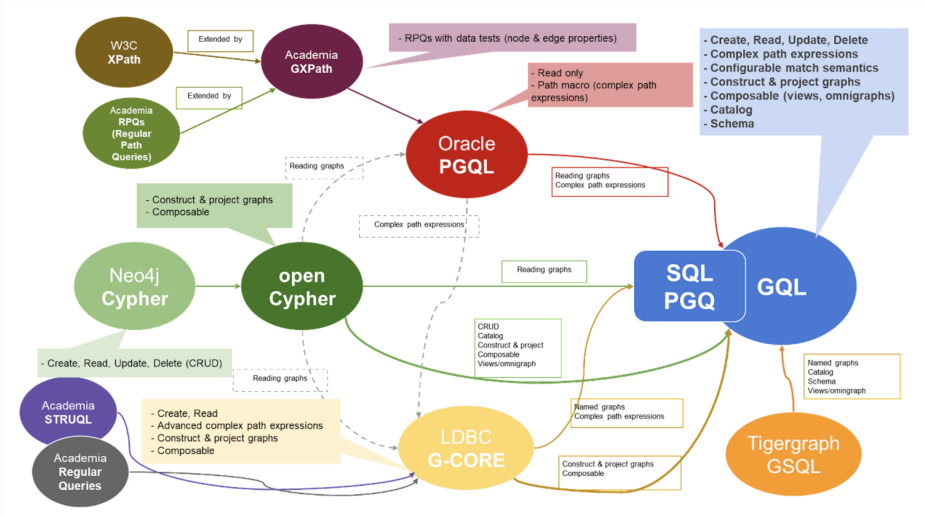
This past June, national standards bodies around the world – belonging to ISO/IEC's Joint Technical Committee 1 (responsible for IT standards) – began voting on the GQL project proposal.

Graph Query Language (GQL) is a new language being developed and maintained by the same international working group that also maintains the SQL standard.

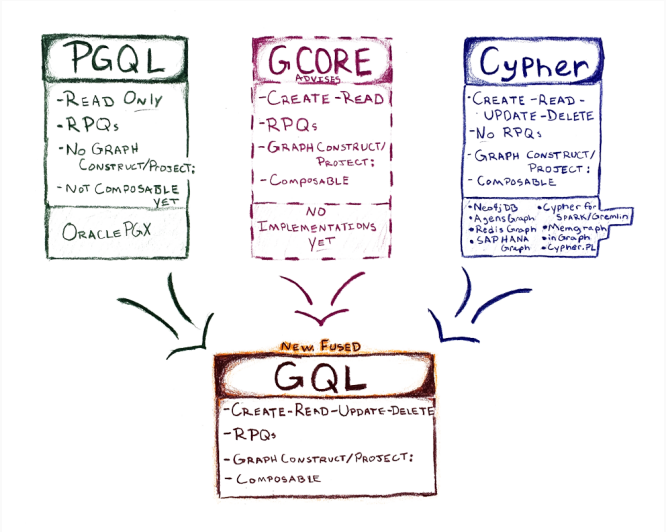
Now, it's official: Earlier this week, **the ballot closed and the proposal passed**, with seven countries putting experts forward to work on the four-year standard query language project.

[1] The very 1st since SQL itself!

GQL


W3C
XPath
Extended by
Academia
GXPath
- RPQs with data tests (node & edge properties)
- Create, Read, Update, Delete
- Complex path expressions
- Configurable match semantics
- Construct & project graphs
- Composable (views, omnigraphs)
- Catalog
- Schema
Extended by
Academia
RPQs
(Regular
Path
Queries)
- Read only
- Path macro (complex path expressions)
Oracle
PGQL
Reading graphs
Reading graphs
Complex path expressions
- Construct & project graphs
- Composable
Complex path expressions
Neo4j
Cypher
open
Cypher
Reading graphs
SQL
PGQ
GQL
CRUD
Catalog
Construct & project
Composable
Views/omnigraph
- Create, Read, Update, Delete (CRUD)
Reading graphs
Named graphs
Catalog
Schema
Views/omnigraph
Academia
STRUQL
Academia
Regular
Queries
- Create, Read
- Advanced complex path expressions
- Construct & project graphs
- Composable
LDBC
G-CORE
Named graphs
Complex path expressions
Construct & project graphs
Composable
Tigergraph
GSQL

GQL (cont'd)


PGQL
-READ ONLY
-RPQs
-NO GRAPH CONSTRUCT/PROJECT
-NOT COMPOSABLE YET
ORACLE PGX
G CORE
ADVISES
-CREATE -READ
-RPQs
-GRAPH CONSTRUCT/PROJECT
-COMPOSABLE
NO IMPLEMENTATIONS YET
Cypher
-CREATE -READ -UPDATE -DELETE
-NO RPQs
-GRAPH CONSTRUCT/PROJECT
-COMPOSABLE
•Neo4j DB
•Agens Graph
•Redis Graph
•SAP HANA Graph
•Cypher for SPARK/Gremlin
•Memgraph
•inGraph
•Cypher.PL
NEW FUSED
GQL
-CREATE-READ-UPDATE-DELETE
-RPQs
-GRAPH CONSTRUCT/PROJECT
-COMPOSABLE