Graph Databases

Guillaume Raschia — Nantes Université

Last update: November 9, 2025

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

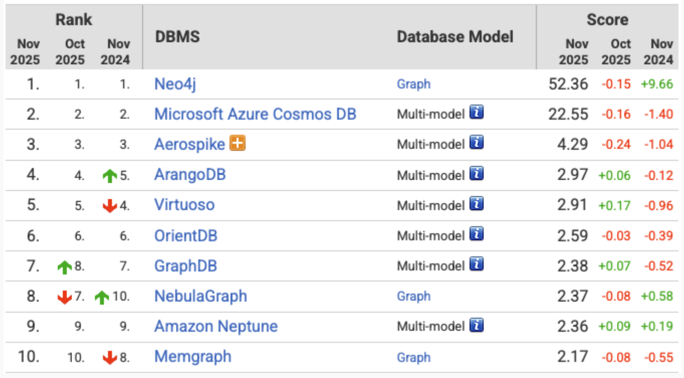
- Join $\bowtie$ is the key operator: costly!
- Reachability: $\bigcup_k R \bowtie_1 R \bowtie_2 \ldots \bowtie_k R$: **Recursion** and **fixpoint**

Graph DB

neo4j

The #1 Database for Connected Data

Graph DB (cont'd)



Rank			DBMS	Database Model	Score		
Nov 2025	Oct 2025	Nov 2024			Nov 2025	Oct 2025	Nov 2024
1.	1.	1.	Neo4j	Graph	52.36	-0.15	+9.66
2.	2.	2.	Microsoft Azure Cosmos DB	Multi-model	22.55	-0.16	-1.40
3.	3.	3.	Aerospike	Multi-model	4.29	-0.24	-1.04
4.	4.	↑5.	ArangoDB	Multi-model	2.97	+0.06	-0.12
5.	5.	↓4.	Virtuoso	Multi-model	2.91	+0.17	-0.96
6.	6.	6.	OrientDB	Multi-model	2.59	-0.03	-0.39
7.	↑8.	7.	GraphDB	Multi-model	2.38	+0.07	-0.52
8.	↓7.	↑10.	NebulaGraph	Graph	2.37	-0.08	+0.58
9.	9.	9.	Amazon Neptune	Multi-model	2.36	+0.09	+0.19
10.	10.	↓8.	Memgraph	Graph	2.17	-0.08	-0.55

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

Graph DB and NoSQL

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

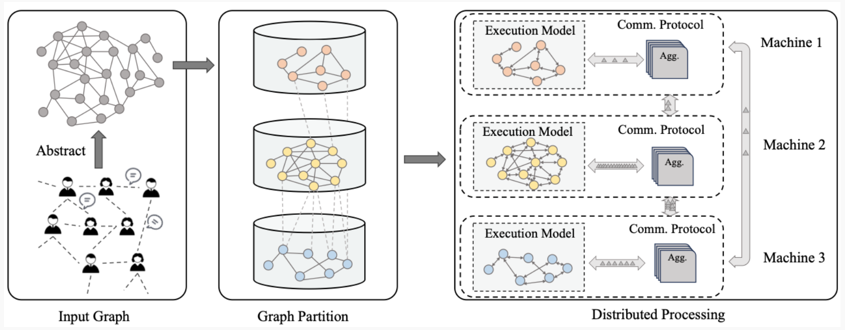
Bronson et al. (2013). *TAO: Facebook's Distributed Data Store for the Social Graph.* USENIX ATC.

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.

Distributed Graph Processing

Challenges: Parallelism • Load balance • Communication • Bandwidth

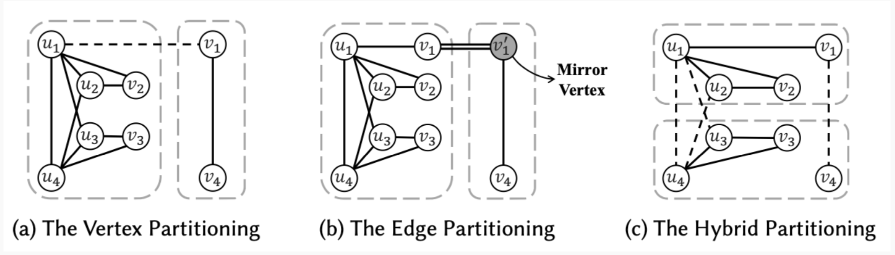


Lingkai Meng, et al. 2024. A Survey of Distributed Graph Algorithms on Massive Graphs. ACM Comput. Surv. 57, 2, Article 27 (February 2025), 39 pages.
https://doi.org/10.1145/3694966

Graph Partitioning

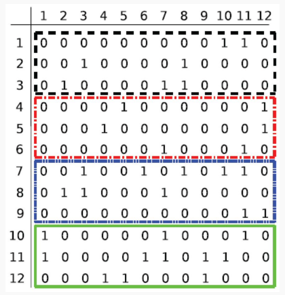
Usually for a particular (set of) operation(s): shortest path, freq. patterns, BFS, ...

Graph partitioning is highly "counter-intuitive"


u_1
u_2
u_3
u_4
v_1
v_2
v_3
v_4
v_1'
Mirror Vertex
(a) The Vertex Partitioning
(b) The Edge Partitioning
(c) The Hybrid Partitioning

1-Dimensional Partitioning

- Partitioning of the adjacency matrix
- Focus on Breadth-First Search, with **order dependency** on the vertices



	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, those that lead to other processors
2. The neighbors of the vertices in F form a set of neighboring vertices N
   - Some owned by the current processor, some by others
3. Messages are sent to all "neighboring 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 (LPG)

A (attributed) multi-relational labeled digraph $G = (N, E, \rho, \lambda, \sigma)$ where

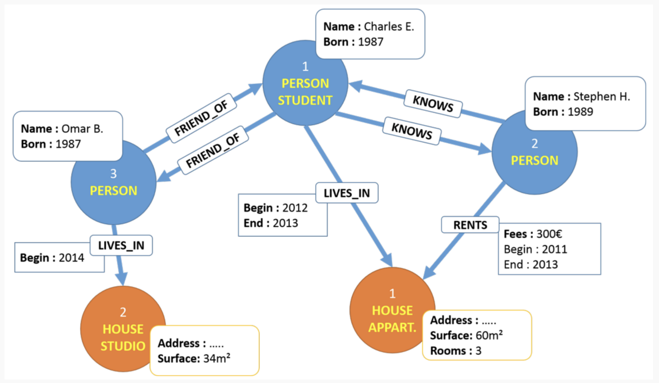
- N is a finite set of nodes (id's)
- E is a finite set of edges (id's), such that $N \cap E = \emptyset$
- $\rho : E \to N \times N$ is the usual incidence function
- $\lambda : (N \cup E) \nrightarrow \Sigma^*$ is a **labeling** partial function for nodes and edges
- $\sigma : (N \cup E) \times K \nrightarrow V$ is a partial function to map key/value pairs—**properties**—to nodes and edges.

Extensions to

hypernodes (nested graphs), hypergraph (with hyperedges to sets of nodes), multigraphs (multiple single-relational edges)

Labeled Property Graph Example

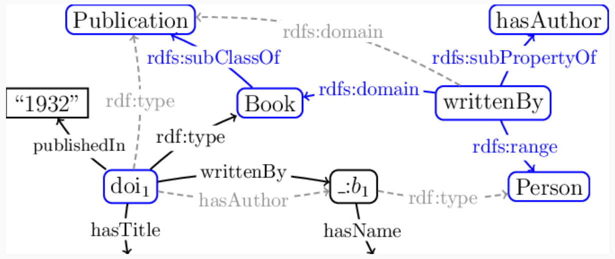
Node labels are "groups" whereas edge labels denote compulsory relationships



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**



S. Cebiric, et al. (2019). Summarizing Semantic Graphs: A Survey. The VLDB Journal. 28. 10.1007/s00778-018-0528-3.

- 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)

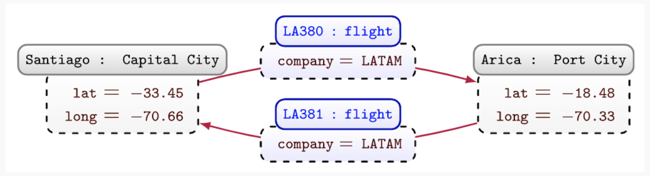
RDF Data Model: Edge-Labeled Digraph

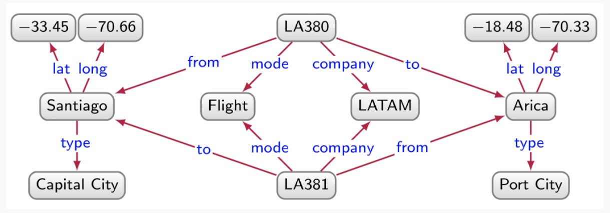
Vertex set V is split into URIs (U), literals (L), and blank/anonymous nodes (B), such that:

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

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

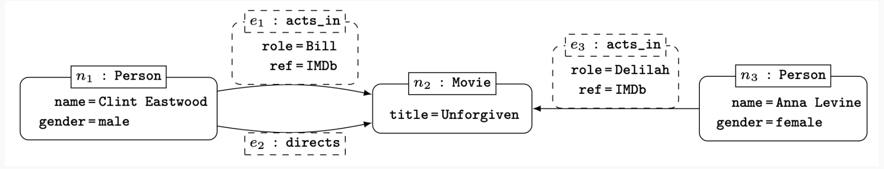
Graph Query Languages: Core Features

Renzo Angles et al. (2017) *Foundations of Modern Query Languages for Graph Databases.* ACM Comp. Surv. 50, 5, Article 68 (Sept. 2018), 40 pages.
https://doi.org/10.1145/3104031

Principles

1. Subgraph Matching
2. Path Expressions

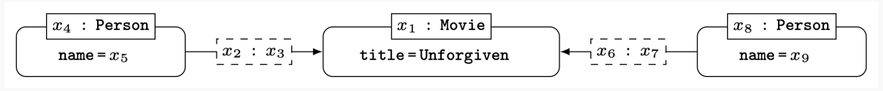
LPG Running Example



Pattern Matching

Basic Graph Patterns (bgp)

- “mimick” equi-selection and (implicit) natural join



Semantics of the Query Q

- subgraph **homomorphism**: multiple variables in Q could map to the same term in the graph G
- subgraph **isomorphism**: bijective mapping
- **bag** or **set** in the result

Pattern Matching (cont’d)

Complex Graph Patterns (cgp)

- bgp extended to include usual relational-like operators such as:
  - **projection**: filter out some matching variables in the output
  - θ-selection (**filter**): the usual way
  - **join**: define two distinct graph patterns Q_1 and Q_2 with shared variables
  - **union**: match pattern Q_1 or pattern Q_2
  - **difference**: negated pattern (tricky)
  - **option**: Q_2 acts as a left-outer join on Q_1, adding its matching wherever it is possible

Navigation

Path Query

- Use paths as a core concept
- most often, **Regular Path Queries**
- dis/liked posts y of friends-of-a-friend of x: $P := x \xrightarrow{\texttt{knows}^+.(\texttt{likes}|\texttt{dislike})} y$
- co-star (x, y) relationship: $P' := x \xrightarrow{\texttt{acts\_in.acts\_in}^-} y$
- Bacon number of x: $P'' :=$ Kevin Bacon $\xrightarrow{(\texttt{acts\_in.acts\_in}^-)^+} y$

Semantics?

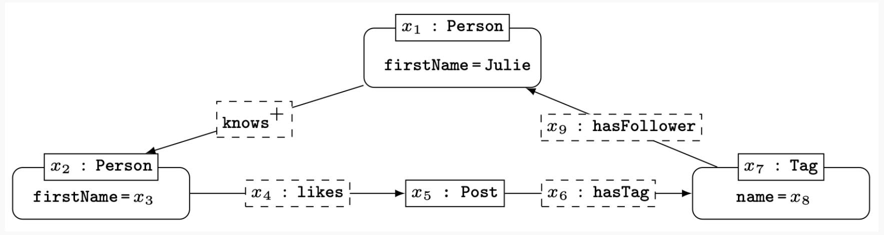
shortest path ● no-repeated node ● no-repeated edge

Navigation (cont'd)

Navigational Graph Patterns (ngp)

- paths incorporated into bgps
- edge labels can be constants, variables, RPQs or the special symbol $*$

"In a social network, find all friends of friends of Julie that liked a post with a tag that Julie follows"



Navigation (cont'd)

Complex Navigational Graph Patterns (cngp)

- ngps extended with operators
  - **project** x_5 to get the "recommended posts" for Julie;
  - duplicate the all pattern to get the posts for Alice, and then use the **union** to get the posts recommended for Julie or Alice,
  - or **intesection** to find posts recommended for both,
  - or **difference** to fin posts recommended for Julie but not Alice,
  - or **filter** dates to find more recent posts
  - ...

Querying Knowledge Graphs

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

Alternatives

- **Native API** in generic programming language

  depends on PL and Graph DB
- **Procedural**—algebraic-based—language

  depends on PL
- **Declarative**—logic-based—language

  depends on Graph DB

Nowadays **GQL is an ISO standard**!

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 (id. next slides): 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();
```

Neo4j Cypher

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
```

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 (?))
    ) ;
```

Cypher Use Case

User story

As an employee,

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

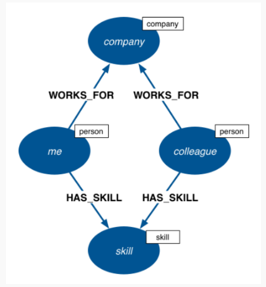
So that we can exchange knowledge

Query

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

Query as a Graph Pattern

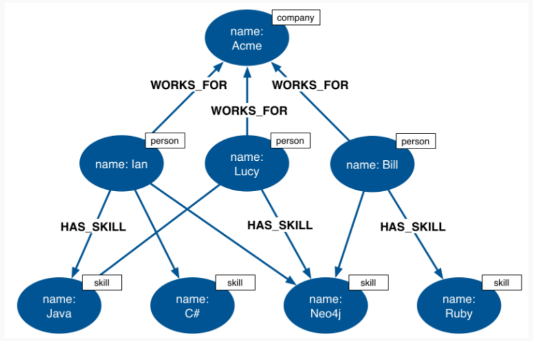
```
MATCH (company)<-[:WORKS_FOR]-(me:person)-[:HAS_SKILL]->(skill),
    (company)<-[:WORKS_FOR]-(colleague)-[:HAS_SKILL]->(skill)
WHERE  me.name= {name}
RETURN colleague.name AS name,
       count(skill) AS score,
       collect(skill.name) AS skills
ORDER BY score DESC
```



Running the Query

Cypher uses isomorphism-based no-repeated-edges bag semantics

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



Regular Paths in Cypher

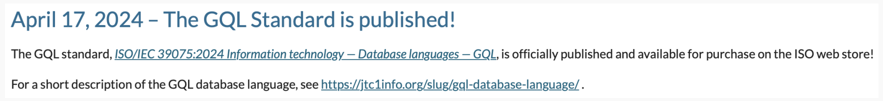
Examples

```
()
(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)
```

A New 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: April 17th, 2024
- A. Deutsch et al. (2022). *Graph Pattern Matching in GQL and SQL/PGQ*. SIGMOD ’22

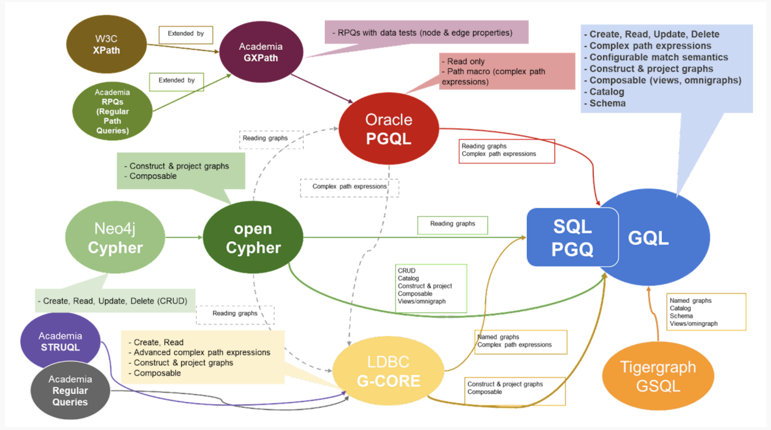

April 17, 2024 – The GQL Standard is published!

The GQL standard, ISO/IEC 39075:2024 Information technology — Database languages — GQL, is officially published and available for purchase on the ISO web store!

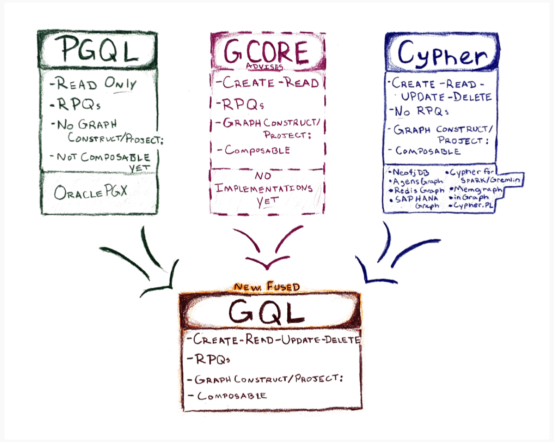
For a short description of the GQL database language, see https://jtc1info.org/slug/gql-database-language/ .

[1] The very 1st since SQL itself!

GQL


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

GQL (cont’d)



References

Maciej Besta, Robert Gerstenberger, Emanuel Peter, Marc Fischer, Michał Podstawski, Claude Barthels, Gustavo Alonso, and Torsten Hoefler. 2023. Demystifying Graph Databases: Analysis and Taxonomy of Data Organization, System Designs, and Graph Queries. ACM Comput. Surv. 56, 2, Article 31 (February 2024), 40 pages. https://doi.org/10.1145/3604932

Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan Reutter, and Domagoj Vrgoč. 2017. Foundations of Modern Query Languages for Graph Databases. ACM Comput. Surv. 50, 5, Article 68 (September 2018), 40 pages. https://doi.org/10.1145/3104031

Alin Deutsch, Nadime Francis, Alastair Green, Keith Hare, Bei Li, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Wim Martens, Jan Michels, Filip Murlak, Stefan Plantikow, Petra Selmer, Oskar van Rest, Hannes Voigt, Domagoj Vrgoč, Mingxi Wu, and Fred Zemke. 2022. Graph Pattern Matching in GQL and SQL/PGQ. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD ’22). Association for Computing Machinery, New York, NY, USA, 2246–2258. https://doi.org/10.1145/3514221.3526057

Lingkai Meng, Yu Shao, Long Yuan, Longbin Lai, Peng Cheng, Xue Li, Wenyuan Yu, Wenjie Zhang, Xuemin Lin, and Jingren Zhou. 2024. A Survey of Distributed Graph Algorithms on Massive Graphs. ACM Comput. Surv. 57, 2, Article 27 (February 2025), 39 pages. https://doi.org/10.1145/3694966