Database	Model	Primary Query Language(s)	Strength
Neo4j	Property Graph	**Cypher**	Developer ergonomics, ecosystem, knowledge graphs.
Oracle Graph	Property Graph / RDF in RDBMS	**SQL/PGQ**, SPARQL	Enterprise RDBMS features + graph.
RushDB	Property Graph (zero-config)	JSON search API (Neo4j underpinnings)	Instant graph from JSON/CSV; rapid prototyping.
TigerGraph	Distributed Property Graph	**GSQL**	Massive parallel analytics, real-time recommendations.
JanusGraph	Property Graph on backends	**Gremlin** (TinkerPop)	Portability & backend flexibility.
RedisGraph	In-memory Property Graph	**Cypher (subset)**	Ultra-low latency, Redis integration.
Dgraph	Native distributed graph	**GraphQL / DQL**	GraphQL-first APIs & scale.
Neptune, ArangoDB, Nebula	Mixed	**SPARQL / Gremlin / AQL / nGQL**	Managed AWS service, multi-model, distributed graph.

Vishal Mysore

Graph Databases & Query Languages in 2025 — A Practical Guide

6 min read · 17 hours ago

Vishal Mysore Follow

Neo4j, Oracle Graph, RushDB, TigerGraph, and other leading graph systems compared for architects and developers

Applications that need relationship-aware intelligence (recommendations, fraud detection, knowledge graphs, graph-native ML, transit systems) increasingly rely on graph databases. But not all graph stores are the same: they differ in data model, scaling architecture, query language, analytic capabilities, and ecosystem. This article summarizes the state of several major graph databases and the query languages they expose — with practical guidance on when to use which.

. . .

Summary

- **Neo4j** — Best for developer productivity, property-graph modeling, rich ecosystem and Cypher query language.
- **Oracle Graph / Oracle Database (Property Graphs + SQL/PGQ)** — Enterprise-grade, brings graph capabilities into a converged RDBMS with SQL/PGQ support for property graphs; great if you need strong transactional RDBMS features plus graph.
- **RushDB** — New "zero-config" / instant graph DB built on Neo4j that auto-normalizes JSON/CSV to a property graph; targets fast developer onboarding for AI/modern apps. Good for prototypes and teams that want minimal schema work.
- **TigerGraph** — Designed for large-scale graph analytics; its proprietary **GSQL** is a high-level, Turing-complete, SQL-like language optimized for parallel graph algorithms. Use it for heavy analytics and production recommendations.
- **Gremlin (TinkerPop) / JanusGraph** — Gremlin is a graph traversal language supported by JanusGraph and many systems — good when portability across backends is required.
- **Dgraph, RedisGraph, Nebula, ArangoDB, Amazon Neptune** — each provides distinctive tradeoffs (native GraphQL or DQL, in-memory speed, multi-model, SPARQL/Gremlin support). Pick by API preferences and scale needs.

. . .

What to compare when choosing a graph database

1. **Data model**: property graph vs RDF (semantic web) vs multi-model
2. **Query language**: expressiveness, standardization, learning curve (Cypher, GSQL, Gremlin, SPARQL, AQL, DQL)
3. **Scale & performance**: single-node vs distributed & parallel analytics

4. **Analytics & graph algorithms**: builtin traversal, path-finding, centrality, community detection
5. **Operational needs:** ACID, backups, cloud offering, enterprise support
6. **Ecosystem**: drivers, ORMs, visualization, integrations with ML/LLMs

. . .

Deep dives: systems & languages

Neo4j — Cypher (declarative property-graph)

What it is: Industry-leading property-graph DB focused on developer ergonomics, ACID transactions, and an expressive declarative graph query language called **Cypher**. Neo4j emphasizes pattern matching and readable queries.

Query language: Cypher — declarative, SQL-like for graphs; now part of GQL discussions and supported by openCypher.

Example:

```
MATCH (p:Patient)-[:HAS_DIAGNOSIS]->(d:Disease {name: "Diabetes"})
RETURN p.name, d.name;
```

When to choose Neo4j: Rapid development, tight tooling (Bloom, Aura cloud, drivers for many languages), strong community, and use cases like knowledge graphs, fraud detection, product graphs.

. . .

Oracle Graph / Property Graphs in Oracle Database — SQL/PGQ & converged platform

What it is: Oracle exposes graph capabilities inside Oracle Database (including "property graphs") and supports **SQL/PGQ** (SQL Property Graph Queries — an ISO/IEC graph query extension) and SPARQL for RDF. This lets you use native SQL tools and enterprise features (security, backup, scale) together with graph analytics.

Query languages supported:

- **SQL/PGQ** — SQL extension for property graph queries (standardizing ways to express graph traversals using SQL)
- **PGQL** on some Oracle tooling / SPARQL for RDF where applicable. Example (conceptual SQL/PGQ):

```
SELECT *
FROM MATCH (p:Person)-[r:FRIEND_OF*1..2]->(q:Person)
WHERE p.id = 'U123';
```

When to choose Oracle Graph: Existing Oracle footprint, need enterprise database features plus graph querying and analytics without deploying a separate graph DB.

. . .

RushDB — Zero-config / Instant Graph (developer-first)

What it is: RushDB is an emergent open-source/ commercial project that promises a zero-config, instant graph database experience: push JSON/CSV, and RushDB auto-normalizes into a labeled meta property graph. It's built with Neo4j under the hood (or integrates with it) and targets fast development for AI applications. Documentation and vendor pages emphasize no-schema ingestion and a JSON search API.

Query surface: RushDB exposes a JSON-centric query/search API (developer-friendly), rather than forcing learning a new graph DSL. Good for teams that want graph benefits without upfront modeling. Example use-case: rapid prototyping of RAG/LLM knowledge graphs from JSON datasets.

When to choose RushDB: Rapid prototyping, projects where incoming data is heterogeneous JSON and you want immediate graph affordances without schema design.

. . .

TigerGraph — GSQL (parallel analytics)

What it is: TigerGraph is designed for **large-scale, distributed** graph analytics with built-in parallelism and performance optimizations for multi-hop traversals and graph algorithms. It targets production analytics, real-time recommendations, and ML feature extraction.

Query language: GSQL — SQL-like, Turing-complete language built for expressing iterative graph algorithms, with strong support for bulk-synchronous parallel execution. Example (snippet concept):

```
CREATE QUERY shortestPath(VERTEX<user> start, VERTEX<user> end) {
  /* GSQL algorithmic logic */
}
```

When to choose TigerGraph: You need industrial-scale analytics, real-time recommendations, or to run complex graph algorithms in production with low latency.

. . .

JanusGraph / Apache TinkerPop — Gremlin (traversal language)

What it is: JanusGraph is an open-source distributed graph DB that often sits on scalable storage backends (Cassandra, HBase). It implements Apache TinkerPop stack so applications can use **Gremlin** — a traversal-based, imperative query API that runs across multiple graph engines.

Query language: Gremlin — a path/traversal DSL (imperative style) that's portable across engines that implement TinkerPop.

Example:

```
g.V().hasLabel('person').has('name','Alice').out('knows').values('name')
```

When to choose Gremlin/JanusGraph: Need portability across multiple storage backends, or an imperative traversal API for complex traversals and programmatic graph processing.

. . .

RedisGraph — Cypher-like in-memory graph

What it is: RedisGraph embeds graph capabilities into Redis using a compact, adjacency-matrix style representation for high-performance in-memory queries. It supports a subset of Cypher. Good for extremely low-latency lookups and transient graph workloads.

When to choose RedisGraph: Ultra-low latency, ephemeral graphs, or when you already use Redis and need fast graph ops.

. . .

Dgraph — DQL / GraphQL-native

What it is: Dgraph is a distributed native graph database that exposes GraphQL as a first-class API and also its proprietary DQL (previously GraphQL+-). It targets developers who prefer GraphQL for graph queries and APIs.

When to choose Dgraph: You want GraphQL-native graph APIs with horizontal scaling and developer-friendly schemas.

. . .

Other notable engines & languages

- **Amazon Neptune:** Managed graph service supporting **SPARQL** (RDF) and **Gremlin** (property graph). Good for AWS-native workloads.
- **ArangoDB:** Multi-model DB (document + graph) using **AQL** (Arango Query Language).
- **NebulaGraph:** Distributed graph DB with **nGQL** (Cypher-like). NebulaGraph

. . .

Quick comparison table

Database	Model	Primary Query Language(s)	Strength
Neo4j	Property Graph	**Cypher**	Developer ergonomics, ecosystem, knowledge graphs.
Oracle Graph	Property Graph / RDF in RDBMS	**SQL/PGQ**, SPARQL	Enterprise RDBMS features + graph.
RushDB	Property Graph (zero-config)	JSON search API (Neo4j underpinnings)	Instant graph from JSON/CSV; rapid prototyping.
TigerGraph	Distributed Property Graph	**GSQL**	Massive parallel analytics, real-time recommendations.
JanusGraph	Property Graph on backends	**Gremlin** (TinkerPop)	Portability & backend flexibility.
RedisGraph	In-memory Property Graph	**Cypher (subset)**	Ultra-low latency, Redis integration.
Dgraph	Native distributed graph	**GraphQL / DQL**	GraphQL-first APIs & scale.
Neptune, ArangoDB, Nebula	Mixed	**SPARQL / Gremlin / AQL / nGQL**	Managed AWS service, multi-model, distributed graph.

Vishal Mysore

. . .

Practical examples — short snippets

Cypher (Neo4j):

```
CREATE (a:Person {name:'Alice'}),(b:Person {name:'Bob'}),
(a)-[:KNOWS]->(b);
MATCH (a:Person {name:'Alice'})-[:KNOWS]->(b) RETURN b.name;
```

Gremlin (JanusGraph/TinkerPop):

```
g.addV('person').property('name','Alice')
g.V().has('name','Alice').out('knows').values('name')
```

GSQL (TigerGraph) — conceptual:

```
CREATE QUERY getFriends(VERTEX<person> p) FOR GRAPH MyGraph {
  start = {p};
  res = SELECT t FROM start:s - (knows:e) -> :t;
  PRINT res;
}
```

SPARQL (RDF example):

```
SELECT ?person WHERE {
  ?person a ex:Person .
  ?person ex:knows ex:Bob .
}
```

DQL / GraphQL (Dgraph):

```
{
  queryPerson(func: eq(name, "Alice")) {
    name
    friend { name }
  }
}
```

How query language affects development & portability

- **Declarative (Cypher, SPARQL, AQL):** Easier to read and maintain; good for business queries and analytics.
- **Traversal/Imperative (Gremlin):** More programmatic, flexible for complex traversals and streaming-like graph processing.
- **Turing-complete (GSQL):** Enables writing complex algorithms directly inside the DB — useful for advanced analytics but ties you to the vendor.
- **GraphQL/DQL:** Great for API-first teams that want a single programmable interface.

If portability matters, prefer **TinkerPop/Gremlin** or standard approaches (SPARQL for RDF). If productivity is key, **Cypher** and **GraphQL** approaches win.

...

Recommendations by use-case

- **Developer prototyping, knowledge graphs, RAG for LLMs:** Neo4j (Cypher) or RushDB for zero-config ingestion and fast turn-up.
- **Large-scale real-time recommendations / heavy graph ML pipelines:** TigerGraph (GSQL) or a distributed system with strong parallelism.
- **Enterprise transactional + graph analytics inside existing RDBMS:** Oracle Graph (SQL/PGQ) to avoid moving data out of the database.
- **Cloud-native, managed option on AWS:** Amazon Neptune (SPARQL + Gremlin).
- **Graph APIs + front-end teams using GraphQL:** Dgraph or ArangoDB. Hypermode
- **Ultra-low-latency in-memory needs:** RedisGraph.

...

Caveats & vendor lock-in

- Vendor-specific languages (GSQL, nGQL) can offer great capabilities but increase lock-in. If you value portability, prefer **open standards** (SPARQL, Gremlin, openCypher/GQL-compliant syntax).
- Newer tools like **RushDB** aim to reduce schema friction — a tradeoff exists between convenience and control/observability.

...

Final thoughts — matching language to problem

Pick the graph database based on:

1. **Your query patterns** (path traversals? analytical algorithms? ad-hoc queries?)

2. **Scale & latency needs** (single-node vs distributed)
3. **Ecosystem fit** (cloud provider, existing RDBMS, drivers for your stack)
4. **Long-term portability or openness** (standards vs proprietary features)

If you're building knowledge graphs to ground LLMs and reduce hallucination, start with Neo4j or RushDB for fast ingestion and Cypher-led querying; for production-grade analytics at scale consider TigerGraph or a distributed graph with Gremlin-compatible tooling for portability.

Follow

Written by Vishal Mysore

755 followers · 4 following

Holder of multiple patents in AI and software engineering. Passionate about building scalable systems, optimizing performance, & driving AI-powered innovation.

Vishal Mysore

What is BMAD-METHOD™? A Simple Guide to the Future of AI-Driven Development

The BMAD-METHOD™ (Breakthrough Method of Agile AI-Driven Development) framework offers a surprisingly simple, yet highly innovative...

Sep 8 187 2

Vishal Mysore

GitHub Spec Kit vs BMAD-Method: A Comprehensive Comparison : Part 1

GitHub’s open-source Spec Kit is a toolkit for spec-driven development that provides a structured process to bring specification-first…

Sep 15 69 4

Vishal Mysore

Comprehensive Guide to Spec-Driven Development Kiro, GitHub Spec Kit, and BMAD-METHOD

What is Spec-Driven Development (SDD)?

Sep 30 16

Vishal Mysore

RAG 2.0 : Advanced Chunking Strategies with Examples.

Text chunking is a critical preprocessing step in RAG Based Large Language Model (LLM) applications. While basic strategies like...

Oct 2 24

See all from Vishal Mysore

Recommended from Medium

In Medialesson by Marius Schröder

JSON vs TOON — A new era of structured input?

Why structure matters more than ever

Nov 3 163 9

In AI Software Engineer by Joe Njenga

Anthropic Just Solved AI Agent Bloat — 150K Tokens Down to 2K (Code Execution With MCP)

Anthropic just released smartest way to build scalable AI agents, cutting token use by 98%, shift from tool calling to MCP code execution

4d ago 417 33

Vishal Mysore

BMAD-METHOD™ Universal Agent Framework: Expansion Packs That Transform AI Into Any Domain

The BMAD-METHOD™ Universal Agent Framework with Expansion Packs is redefining how AI frameworks scale across industries. Unlike traditional...

Sep 23 8

CodeOrbit

I Built a RAG System for 100,000 Documents — Here's the Architecture

My production system crashed at 2 AM because I underestimated vector databases.

Nov 1 583 15

In Towards AI by Devashish Datt Mamgain

Pseudo-Knowledge Graphs for Better RAG

Retrieval-Augmented Generation (RAG) was supposed to give Large Language Models perfect memory: ask a question, fetch the exact facts, and...

Oct 30 35

Arvind Kumar

Building a Ticketing System: Concurrency, Locks, and Race Conditions

What happens when 100,000 fans try to book the same concert ticket at exactly 10:00 AM? Let's design a ticketing system that prevents...

Oct 30 616 8

See more recommendations