GraphRAG

By 

Graph DB

A database built on **graph theory**

Data is stored as:

- **Nodes** → entities
- **Relationships (verbs)** → how entities connect
- **Properties** → attributes

Ex:

Nodes

Represent entities

Examples: Person, Movie, Product

Relationships

Connect two nodes

Examples: FOLLOWS, PURCHASED, LIKES

Properties

Key-value attributes

Examples: name: "Alice", age: 24

Nodes: Alice, Bob, Movie

Relationships:

Alice → FRIEND_OF → Bob

Bob → LIKES → Movie

Graph DB

Feature	Relational DB	Graph DB
Data model	Tables, Rows	Nodes, Relationships
Joins	Expensive	No joins; direct traversal
Schema	Rigid	Flexible
Deep query	Slow	Fast
Best for	Structured tabular	Connected data

- **Neo4j** is World's most popular **native graph database**
- Iternatives are **PuppyGraph, AWS Neptune, TigerGraph, OrientDB, and ArangoDB**
- Uses **Cypher Query Language**
- Visual graph exploration
- Offers AuraDB (cloud) for easy deployment

Neo4j Editions

Feature / Edition	Community	Desktop	Enterprise	Aura (Cloud)
Summary	Free basic Neo4j for small/local use	Enterprise features for free (dev only)	Full production-grade Neo4j	Fully managed cloud Neo4j
Cost	Free	Free	Paid	Free → Paid **Aura Free** (small) **Aura Professional** **Aura Enterprise**
Usage	Local + small production	Local development only	Production systems	Cloud-hosted apps
Clustering	❌ No	❌ No	✔️ Yes	✔️ Yes
Security (RBAC)	❌ No	✔️ Yes (dev only)	✔️ Full	✔️ Full
Backups	Basic	Limited	✔️ Hot backups	✔️ Automatic backups
GUI	Neo4j Browser	Desktop App + Neo4j Browser	Neo4j Browser + tools	Web Console + Neo4j Browser
Cloud Managed	❌ No	❌ No	❌ No	✔️ Yes
Best For	Learning, small apps	Developers, POCs	Large-scale production	Cloud apps, effortless deployment

Neo4j Architecture

- **Graph Engine**
  - The **Graph Engine** is the core part of Neo4j that stores and processes graph data.
- **Bolt Protocol** for fast access
  - **Bolt** is Neo4j's **binary client–server protocol**.
    It is similar to JDBC but optimized specifically for graph queries, Provides fast, low-latency communication between your app and Neo4j
- **Write-Ahead Logging (WAL)**
  - A **Write-Ahead Log** is a file Neo4j writes **before** applying changes to the database and Only then does it update the graph store
- **Clustering**: Causal clustering
  - Neo4j's **distributed cluster architecture** that keeps data consistent across multiple machines.
    Roles in the cluster
    - **Leader** → handles all writes
    - **Followers** → replicate data from leader & can handle reads
    - **Read Replicas** → scale out for massive read workloads

- **APOC(**Awesome Procedures On Cypher):
  - A huge library of **extensions** that provide advanced graph utilities and algorithms.
  - Data cleaning, Graph algorithms (similarity, pagerank, clustering), Import/export utilities, Text and date functions, ETL (Extract–Transform–Load) functions, Procedures to call external APIs. Graph transformations.

Cyper Query Language

Cypher Basics

1. Create a Node - Creates a Person node with a name property:

CREATE (p:Person {name: “Raj"}) or MERGE (p:Person {name: "Raj"});

- Person is a Node label
- p is alias for Person node
- {name:“Raj"} is a property

- CREATE is used when you *always want a new node*.
- MERGE is used when you want a *unique node* (no duplicates).

2. Create a Relationship
Connects two existing people with a FRIEND_OF relationship:

```
MATCH (a:Person {name:"Alice"}), (b:Person {name:"Bob"})
CREATE (a)-[:FRIEND_OF]->(b)
```

- MATCH finds nodes
- CREATE adds the relationship
- Relationships have direction (→)

Cyper Query Language

2. Create a Relationship
Connects two existing people with a FRIEND_OF relationship:

```
CREATE (a:Person {name:"Alice"}), (b:Person {name:"Bob"})
CREATE (a)-[:FRIEND_OF]->(b)
```

- CREATE here create nodes and adds the relationship
- Relationships have direction (→)

Pattern	Meaning	Direction?	Used in CREATE?
() -[]- ()	Match relationship in ANY direction	No	❌ No
() -[]-> ()	Outgoing relationship	Yes	✔ Yes
() <-[]- ()	Incoming relationship	Yes	✔ Yes

Cypher Query Language

3. Query
Find all friends of each person:

```
MATCH (p:Person)-[h:FRIEND_OF]->(friend)
RETURN p,h, friend
```

- MATCH pattern-searches the graph
- Arrow means “connected by a FRIEND_OF relationship”
- Returns pairs of people and their friends

4. Query all movies an actor acted in

```
MATCH (p:Person {name: "Tom Hanks"})-[:ACTED_IN]->(m:Movie)
RETURN m.title
```

- Finds the node for Tom Hanks
- Follows ACTED_IN relationships to movie nodes
- Returns titles of connected movies

```
MATCH p=()-[:DIRECTED]->()
RETURN p LIMIT 25;
```

- Matches **any two nodes** connected by a **DIRECTED** relationship

Cypher Query Language

3. Update data

```
MATCH (p:Person {name:"Bob"})
SET p.age = 26
RETURN p;
```

4. Delete data

```
MATCH (p:Person {name:"Alice"})
DETACH DELETE p;
```

Thank you