🧠 Graph Databases & Cypher Language

Generative AI Bootcamp – Week 2 (Databases & Data Pipelines)

Day 5 – Friday, November 28, 2025

Why Graphs Matter for AI

- Many AI tasks rely on **relationships**: entities, facts, provenance, context
- Graph DBs represent data as **nodes** and **edges**, ideal for:
  - Knowledge graphs
  - Entity linking
  - Reasoning and retrieval augmentation (RAG)
  - Explainability and traceability

Core Graph Concepts

Concept	Description	Example
Node	Entity/object	Person, Paper, Company
Relationship	Connection between nodes	(:Author)-[:WROTE]->(:Paper)
Property	Key-value data on nodes/edges	name, title, date
Label	Node type tag	:Person, :Paper
Cypher	Query language for graphs	`MATCH (a:Author)-[:WROTE]->(p:Paper)`

Cypher Syntax Overview

Operation	Example	Description
CREATE	`CREATE (a:Author {name:'Ada'})`	Add a node
RELATION	`CREATE (a)-[:WROTE]->(p)`	Connect nodes
MATCH	`MATCH (a:Author)-[:WROTE]->(p)`	Query pattern
WHERE	`WHERE p.year > 2020`	Filter
RETURN	`RETURN a.name, p.title`	Output results

Spin Up Neo4j (Docker)

```
docker run -d --name neo4j \\
  -p7474:7474 -p7687:7687 \\
  -e NEO4J_AUTH=neo4j/testpass \\
  -v $PWD/neo4j_data:/data \\
  neo4j:5
```

Access Browser: http://localhost:7474

Login: `neo4j / testpass`

Python Setup

Install dependencies:

```
pip install neo4j
```

Connect from Python using the Bolt protocol:

```python
from neo4j import GraphDatabase
driver = GraphDatabase.driver('bolt://localhost:7687', auth=('neo4j','testpass'))
```

Example Graph: Authors & Papers

```
CREATE (a1:Author {name:'Ada'})
CREATE (a2:Author {name:'Turing'})
CREATE (p1:Paper {title:'AI Ethics', year:2025})
CREATE (a1)-[:WROTE]->(p1)
CREATE (a2)-[:CITED]->(p1);
```

Querying the Graph

```
MATCH (a:Author)-[:WROTE]->(p:Paper)
RETURN a.name, p.title;
```

Pattern Matching reveals relationships easily — no joins required!

Example: Expanding the Graph

```
MATCH (a:Author {name:'Ada'})-[:WROTE]->(p:Paper)
CREATE (p)-[:CITED_BY]->(:Paper {title:'Trustworthy AI', year:2026});
```

Directed edges and **types** make graphs semantically rich.

Why Graphs for LLMs?

- Represent **knowledge structures** behind text corpora
- Enable **semantic retrieval** and **relationship reasoning**
- Combine with **vector stores** for hybrid RAG systems
- Support **explainable AI** via traceable node paths

Graph + Vector = Hybrid Retrieval

Component	Function
Vector Store	Semantic similarity search
Graph DB	Relationship-based reasoning
Combined	Rich, contextual retrieval for agents

Summary

- Graph DBs model relationships naturally
- Cypher = expressive query language
- Great for **knowledge graphs**, **reasoning**, and **explainability**
- Next: Build & query a mini knowledge graph (Neo4j Lab)

🧩 Graph Data Science (GDS) Primer

Optional Extension – Exploring Analytics & AI on Graphs

🧠 What Is Graph Data Science?

- Uses **graph algorithms** to analyze patterns, influence, and clusters
- Focuses on **relationships** rather than isolated data points
- Key applications:
  - Recommender systems
  - Fraud detection
  - Knowledge graph enrichment
  - AI reasoning & explainability

🧮 GDS in Neo4j

Neo4j's **Graph Data Science Library (GDS)** provides over 65 algorithms for:

Category	Example Algorithms	Use Case
Centrality	PageRank, Betweenness	Find key influencers
Community	Louvain, Label Propagation	Detect clusters or topics
Similarity	Jaccard, Node Similarity	Recommend similar entities
Pathfinding	Dijkstra, A*	Discover shortest or critical paths

⚙️ Setup for GDS

If using Docker, start Neo4j Enterprise (GDS preinstalled):

```
docker run -d --name neo4j-gds -p7474:7474 -p7687:7687 \\
 -e NEO4J_AUTH=neo4j/testpass neo4j:5-enterprise
```

Access: http://localhost:7474
Login: `neo4j / testpass`

🧩 Step 1: Project a Graph

GDS works on *in-memory projections* for faster computation.

```
CALL gds.graph.project(
  'myGraph',
  ['Author','Paper'],
  'WROTE'
);
```

🧩 Step 2: Run Algorithms

🏆 PageRank (Node Importance)

```
CALL gds.pageRank.stream('myGraph')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC
LIMIT 5;
```

Finds the most **influential authors or papers** in your citation graph.

🧭 Community Detection (Louvain)

```
CALL gds.louvain.stream('myGraph')
YIELD nodeId, communityId
RETURN gds.util.asNode(nodeId).name, communityId
ORDER BY communityId;
```

Identifies **clusters of related nodes** — e.g., topic communities.

🔍 Node Similarity

```
CALL gds.nodeSimilarity.stream('myGraph')
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).name AS A,
       gds.util.asNode(node2).name AS B,
       similarity
ORDER BY similarity DESC
LIMIT 10;
```

Finds **authors with overlapping connections** → potential collaboration or recommendation.

🧩 Step 3: Graph ML & Embeddings

Generate **node embeddings** for downstream ML or AI pipelines:

```
CALL gds.fastRP.stream('myGraph')
YIELD nodeId, embedding
RETURN gds.util.asNode(nodeId).name AS name, embedding[0..3] AS sample_embedding;
```

Use embeddings for:

- Node classification
- Link prediction
- Hybrid retrieval (Graph + Vector)

🧠 Why GDS Matters for AI

- Adds **structural intelligence** to your data
- Supports **ranking**, **clustering**, and **context reasoning**
- Bridges **symbolic graphs** and **neural embeddings**
- Enables **knowledge-aware agents** and **explainable AI**

🏁 Summary – Graph Data Science

- Graph algorithms = insights from relationships
- GDS provides ready-to-use **analytics** & **ML** primitives
- Powerful for **recommendations**, **fraud detection**, and **RAG reasoning**
- Next step: integrate GDS results with **LLMs and embeddings**