

DS 4300

Neo4j

Mark Fontenot, PhD
Northeastern University

- A Graph Database System that supports both transactional and analytical processing of graph-based data
 - Relatively new class of no-sql DBs
- Considered schema optional (one can be imposed)
- Supports various types of indexing
- ACID compliant
- Supports distributed computing
- Similar: Microsoft CosmoDB, Amazon Neptune

Neo4j - Query Language and Plugins

- Cypher
 - Neo4j's graph query language created in 2011
 - Goal: SQL-equivalent language for graph databases
 - Provides a visual way of matching patterns and relationships
(nodes) - [:CONNECT_TO] -> (otherNodes)
- APOC Plugin
 - Awesome Procedures on Cypher
 - Add-on library that provides hundreds of procedures and functions
- Graph Data Science Plugin
 - provides efficient implementations of common graph algorithms (like the ones we talked about yesterday)

Neo4j in Docker Compose

Docker Compose

- Supports **multi-container management**.
- Set-up is declarative - using YAML `docker-compose.yaml` file
 - services
 - volumes
 - networks, etc.
- 1 command can be used to start, stop, or scale a number of services at one time.
- Provides a consistent method for producing an identical environment (no more “well... it works on my machine!”)
- Interaction is mostly via command line

docker-compose.yaml

services:

neo4j:

container_name: neo4j

image: neo4j:latest

ports:

- 7474:7474
- 7687:7687


environment:

- NEO4J_AUTH=neo4j/\${NEO4J_PASSWORD}
- NEO4J_apoc_export_file_enabled=true
- NEO4J_apoc_import_file_enabled=true
- NEO4J_apoc_import_file_use__neo4j__config=true
- NEO4J_PLUGINS=["apoc", "graph-data-science"]

volumes:

- ./neo4j_db/data:/data
- ./neo4j_db/logs:/logs
- ./neo4j_db/import:/var/lib/neo4j/import
- ./neo4j_db/plugins:/plugins

Never put “secrets” in a
docker compose file.
Use .env files. *(never
sync .env files to GitHub)



*add .env files to .gitignore FIRST

.env Files

- **.env** files - stores a collection of environment variables
- good way to keep environment variables for different platforms separate
 - .env.local
 - .env.dev
 - .env.prod

.env file

```
NEO4J_PASSWORD=abc123!!!
```

Docker Compose Commands

- To test if you have Docker CLI properly installed, run: `docker --version`
- Major Docker Commands
 - `docker compose up` *(containers don't exist (only have image and compose files))
 - `docker compose up -d` *(run detach mode to not see logs in terminal (once you're confident everything is built/running correctly))
 - `docker compose down`
 - `docker compose start` *(containers exist (had done docker compose up in the past but had turned them off through stop); not creating containers from scratch like in up))
 - `docker compose stop`
 - `docker compose build` *(create image before spinning up)
 - `docker compose build --no-cache` *(redownload all layers from scratch to build)

*containers will go up and down much more frequently than a server; why we use docker-compose to specify everything so you can restart the container with all the data still there if something happens

\$

Database access not available. Please use `:server connect` to establish connection. There's a graph waiting for you.

\$:server connect

Connect to Neo4j

Database access might require an authenticated connection

Connect URL - Could not reach Neo4j. `:debug`

bolt://

localhost:7687

Authentication type

Username / Password

Username

neo4j

Password

.....

Connect

9

localhost:7474 Then login.

The screenshot displays the Neo4j Browser interface with the following components and labels:

- Database:** Points to the 'neo4j' database selector in the top left.
- Favorites:** Points to the star icon in the top left sidebar.
- Guides:** Points to the play button icon in the top left sidebar.
- Sidebar:** Points to the left sidebar containing navigation icons.
- Help & resources:** Points to the question mark icon in the bottom left sidebar.
- Browser sync:** Points to the sync icon in the bottom left sidebar.
- Browser settings:** Points to the gear icon in the bottom left sidebar.
- About Neo4j:** Points to the Neo4j logo in the bottom left sidebar.
- Cypher editor:** Points to the query input field at the top.
- Reusable result frame:** Points to the query input field.
- Run query:** Points to the blue play button icon.
- Full screen editor:** Points to the full screen icon.
- Result frame views:** Points to the view toggle buttons (Graph, Table, Text, Code).
- Full screen result frame:** Points to the full screen icon in the top right of the result frame.
- Export:** Points to the export icon in the top right of the result frame.
- Collapse:** Points to the collapse icon in the top right of the result frame.
- Save as Favorite:** Points to the star icon in the top right of the result frame.
- Pin at top:** Points to the pin icon in the top right of the result frame.
- Rerun a query:** Points to the refresh icon in the top right of the result frame.
- Node Properties display:** Points to the 'Overview' panel on the right.
- Zoom in:** Points to the magnifying glass with a plus icon.
- Zoom out:** Points to the magnifying glass with a minus icon.
- Fit to screen:** Points to the square icon.

The Cypher query shown is:

```
neo4j$ MATCH (p:Person {name:'Tom Hanks'})-[]-(m:Movie) RETURN p,m
```

The graph result shows Tom Hanks as a central node connected to various movies. The 'Overview' panel on the right displays:

- Node labels:** * (25), Person (13), Movie (12)
- Relationship Types:** * (25), ACTED_IN (21), DIRECTED (4)
- Displaying 19 nodes, 19 relationships.**

Inserting Data by Creating Nodes

```
CREATE (:User {name: "Alice", birthPlace: "Paris"})
CREATE (:User {name: "Bob", birthPlace: "London"})
CREATE (:User {name: "Carol", birthPlace: "London"})
CREATE (:User {name: "Dave", birthPlace: "London"})
CREATE (:User {name: "Eve", birthPlace: "Rome"})
```

The diagram shows the statement `CREATE (john:User {name: "John"})` with three colored brackets underneath. A blue bracket under `(john)` points to the text **Variable name** (used to reference this node later in the query). A red bracket under `:User` points to the text **Node label** (differentiate different types of nodes). A green bracket under `{name: "John"}` points to the text **Properties** (attached to the node).

```
CREATE (john:User {name: "John"})
```

Variable name
(used to reference this node later in the query)

Node label
(differentiate different types of nodes)

Properties
(attached to the node)

Adding an Edge with No Variable Names


```
CREATE (:User {name: "Alice", birthPlace: "Paris"})
```

```
CREATE (:User {name: "Bob", birthPlace: "London"})
```

```
MATCH (alice:User {name:"Alice"})
```

```
MATCH (bob:User {name: "Bob"})
```

```
CREATE (alice)-[:KNOWS {since: "2022-12-01"}]->(bob)
```



Note: Relationships are directed in neo4j.

Matching

Which users were born in London?

```
MATCH (usr:User {birthPlace: "London"})
```

```
RETURN usr.name, usr.birthPlace
```

usr.name	usr.birthPlace
"Bob"	"London"
"Carol"	"London"
"Dave"	"London"

Download Dataset and Move to Import Folder

Clone this repo:

<https://github.com/PacktPublishing/Graph-Data-Science-with-Neo4j>

In **Chapter02/data** of data repo, unzip the **netflix.zip** file

Copy **netflix_titles.csv** into the following folder where
you put your docker compose file
neo4j_db/neo4j_db/import

Importing Data

Basic Data Importing

Type the following into the Cypher Editor in Neo4j Browser

```
LOAD CSV WITH HEADERS
FROM 'file:///netflix_titles.csv' AS line
CREATE(:Movie {
    id: line.show_id,
    title: line.title,
    releaseYear: line.release_year
})
```


Loading CSVs - General Syntax

```
LOAD CSV
```

```
[WITH HEADERS]
```

```
FROM 'file:///file_in_import_folder.csv'
```

```
AS line
```

```
[FIELDTERMINATOR ',']
```

```
// do stuffs with 'line'
```

Importing with Directors this Time

```
LOAD CSV WITH HEADERS
```

```
FROM 'file:///netflix_titles.csv' AS line
```

```
WITH split(line.director, ",") as directors_list
```

```
UNWIND directors_list AS director_name
```

```
CREATE (:Person {name: trim(director_name)})
```

But this generates duplicate Person nodes (a director can direct more than 1 movie)

Importing with Directors Merged

MATCH (p:Person) DELETE p

LOAD CSV WITH HEADERS

FROM 'file:///netflix_titles.csv' AS line

WITH split(line.director, ",") as directors_list

UNWIND directors_list AS director_name

MERGE (:Person {name: director_name})

Adding Edges

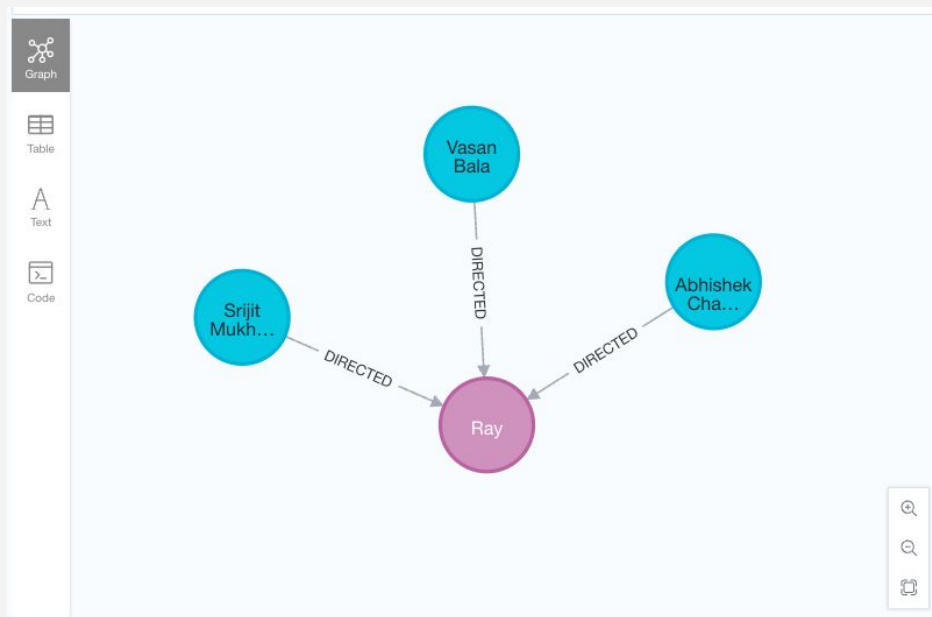
```
LOAD CSV WITH HEADERS
FROM 'file:///netflix_titles.csv' AS line
MATCH (m:Movie {id: line.show_id})
WITH m, split(line.director, ",") as directors_list
UNWIND directors_list AS director_name
MATCH (p:Person {name: director_name})
CREATE (p)-[:DIRECTED]->(m)
```

Gut Check

Let's check the movie titled Ray:

```
MATCH (m:Movie {title: "Ray"})<-[:DIRECTED]-(p:Person)
```

```
RETURN m, p
```



??