Neo4j – A Graph Database System

What is Neo4j?

Neo4j is a graph database system that supports both transactional and analytical processing of graph-based data. It belongs to the NoSQL database family and is used to efficiently store and query data with complex relationships.

Key Features of Neo4j:

- Schema-optional: Does not require a predefined schema but allows one if needed.
- Supports indexing: Improves query performance by creating indexes on node properties.
- ACID compliant: Ensures data integrity by following Atomicity, Consistency, Isolation, and Durability principles.
- Distributed computing support: Scales horizontally across multiple machines.
- Popular alternatives: Microsoft Cosmos DB, Amazon Neptune.

Neo4j Query Language – Cypher

Cypher is Neo4j's graph query language, introduced in 2011. It provides a structured and visual way to query graph databases, similar to SQL for relational databases.

Basic Cypher Syntax:

Nodes are enclosed in parentheses:
(n:Label)

Relationships are represented with -[:RELATIONSHIP]->:

(person)-[:FRIEND_WITH]->(otherPerson)

Example:
Create a node representing a user named Alice:

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

Add a relationship between Alice and Bob:

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

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

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

Note: Relationships in Neo4j are directed.

Querying Data in Neo4j

Find all users born in London:

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

RETURN usr.name, usr.birthPlace
```

Neo4j Plugins

APOC (Awesome Procedures on Cypher) Plugin

- An add-on library providing hundreds of procedures and functions.
- Enhances data import/export, graph algorithms, and query optimizations.

Graph Data Science Plugin

- Provides efficient implementations of graph algorithms like shortest path, centrality, and community detection.

Running Neo4j with Docker Compose

Docker Compose is a tool for managing multiple containers, making it easier to set up Neo4j in a reproducible way.

Basic docker-compose.yaml Configuration for Neo4j:

yaml

CopyEdit

```
services:

  neo4j:

    container_name: neo4j

    image: neo4j:latest

    ports:
```

```
    - 7474:7474

    - 7687:7687

  environment:

    - NEO4J_AUTH=neo4j/${NEO4J_PASSWORD}

    - NEO4J_PLUGINS=["apoc", "graph-data-science"]

  volumes:

    - ./neo4j_db/data:/data

    - ./neo4j_db/import:/var/lib/neo4j/import
```

Important: Never store secrets (e.g., passwords) directly in docker-compose.yaml. Instead, use .env files to store environment variables.

Docker Commands for Neo4j:

Start Neo4j:
```
docker compose up -d
```

Stop Neo4j:
```
docker compose down
```

Rebuild without cache:
```
docker compose build --no-cache
```

Working with Neo4j Browser

Access the Neo4j Browser at localhost:7474, where you can enter Cypher queries.

Example: Checking movies directed by a person:

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

RETURN m, p
```

Importing Data into Neo4j

Downloading a Dataset

To practice with real data, download a dataset:

Clone the dataset repository:
git clone https://github.com/PacktPublishing/Graph-Data-Science-with-Neo4j

1.
2. Extract netflix_titles.csv and place it in the neo4j_db/import folder.

Basic CSV Import in Neo4j

Load a CSV file and create Movie nodes:

cypher

CopyEdit

```
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 CSV Data – General Syntax

cypher

CopyEdit

```
LOAD CSV [WITH HEADERS] FROM 'file:///file_in_import_folder.csv' AS line

[FIELDTERMINATOR ',']  // Specify delimiter if needed

// Perform operations using 'line'
```

Importing Directors and Avoiding Duplicates

Basic approach (creates duplicate nodes):

```
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)})
```

Better approach (avoids duplicates using MERGE):

```
MATCH (p:Person) DELETE p  // Remove existing nodes

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})
```

Creating Relationships (Edges) Between Nodes

Create a relationship between Person and Movie:

```
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)
```

Neo4j Use Cases

Neo4j is widely used in applications where relationships between data points are crucial.

Common Use Cases:

- Social Networks: Friendships and interactions between users.

- Recommendation Systems: Suggesting movies, books, or products based on user preferences.
- Fraud Detection: Identifying fraudulent transactions by analyzing transaction relationships.
- Network and IT Management: Optimizing routing in computer networks.

Summary

- Neo4j is a powerful graph database for storing and querying highly connected data.
- Cypher is Neo4j's query language, allowing SQL-like queries for graphs.
- Docker Compose simplifies Neo4j deployment, and environment variables should be managed via .env files.
- CSV data can be imported into Neo4j, with proper handling to avoid duplicate nodes.
- Neo4j is widely used in social networks, recommendation engines, fraud detection, and IT management.