Exploring Relationships Between Movies and Persons in a Neo4j Graph Database

Jorge Gonçalves (1210107)

Index

Project overview

This project aims to explore the relationships between movies and persons in a Neo4j graph database using the neo4j-driver JavaScript library, the express framework, and the react framework. The application allows users to visualize the relationships between movies and persons, search for persons or movies, and see all the movies that a person has acted in or all the persons that have acted in a particular movie. The data is loaded from a cypher file when the backend is started.

Project functionalities

The project provides the following functionalities:

- View all movies with all the cast.
- View all persons and the movies that they worked on
- Search by a person name and get all the persons with that name and all the movies they worked on
- Search by a movie name and get all the cast

Team Members

This project was developed solely by me, Jorge Gonçalves. I was responsible for all aspects of the project, including the data modeling, query development, application design, frontend development, and backend development. I also conducted all the profiling and testing of the project.

Data Schema

The data schema describes the structure of the data in our application. It consists of classes that represent the entities in our system, along with their attributes and relationships. The relationships between classes are represented by arrows, which indicate that one class is associated with another class.

In this section, we will define the data schema for our movie application. We will start by defining the classes for movies and people. Then, we will define the relationships between these classes.

Movie Class

The Movie class represents a movie. It has the following attributes:

- title: The title of the movie
- tagline: The tagline of the movie
- released: The date the movie was released

Person Class

The Person class represents a person. It has the following attributes:

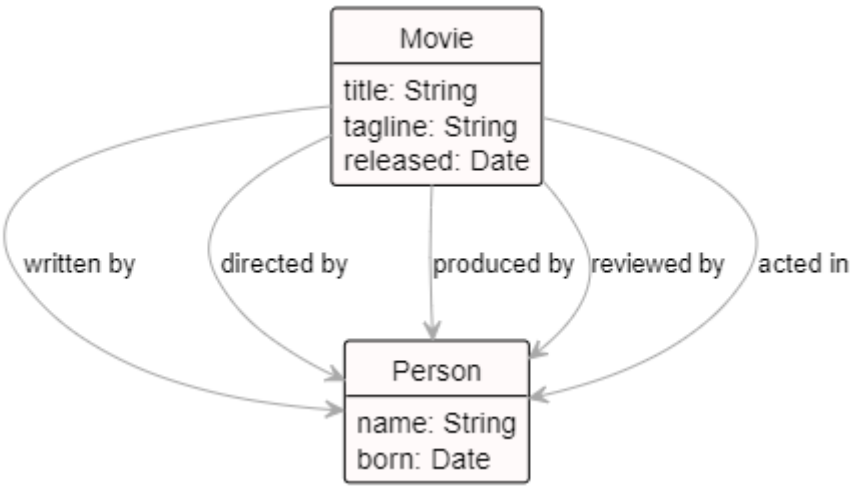
- name: The name of the person
- born: The date the person was born

Relationships

There are several relationships between the Movie and Person classes. Here are some of them:

- A movie can be directed by multiple people.
- A movie can be written by multiple people.
- A movie can be produced by multiple people.
- A movie can be acted in by multiple people.
- A movie can be reviewed by multiple people.

These relationships are represented by arrows in the class diagram. For example, the arrow from the Movie class to the Person class indicates that a movie can be directed by multiple people. The arrow from the Movie class to the Person class indicates that a movie can be acted in by multiple people.



Queries:

In this section, we will explore the queries that were developed to interact with the Neo4j database. Each query will be accompanied by a brief explanation of its functionality and a screenshot of its output, providing insights into the data. All the queries are net using the neo4j driver for JavaScript and converted to Json, to be visualized in the frontend app.

Query 1: Finding Movies and Their Cast

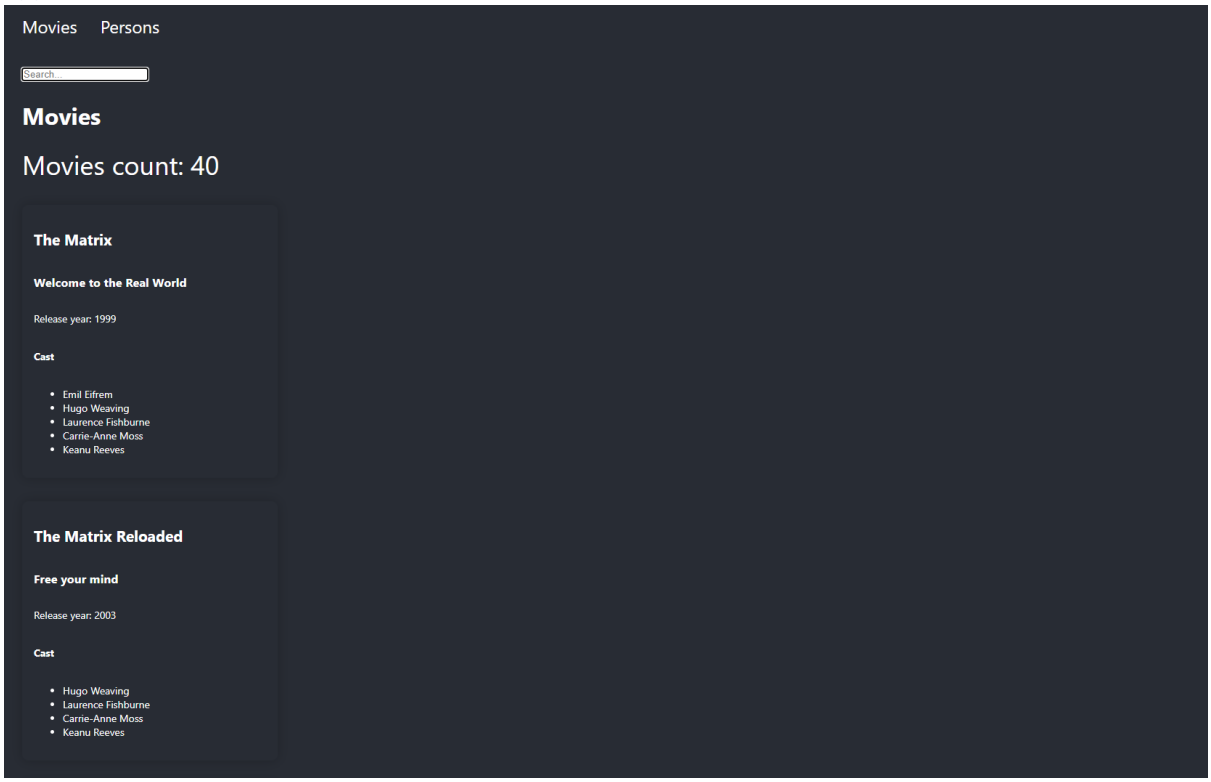
- MATCH (movie:Movie)<-[:ACTED_IN]-(actor:Person) RETURN movie, collect(actor) AS cast

Functionality

This query retrieves a list of movies along with their cast members. It utilizes the ACTED_IN relationship to link movies to actors, effectively traversing the graph to gather information about the ensemble cast of each movie.

Results

App



Neo4j


neo4j$ MATCH (movie:Movie)<-[:ACTED_IN]-(actor:Person) RETURN movie, collect(actor) AS cast;
Overview
Node labels
Relationship types
Displaying 150 nodes, 0 relationships.

Query 2: Searching for Movies by Title

- MATCH (movie:Movie) WHERE movie.title CONTAINS $title RETURN movie

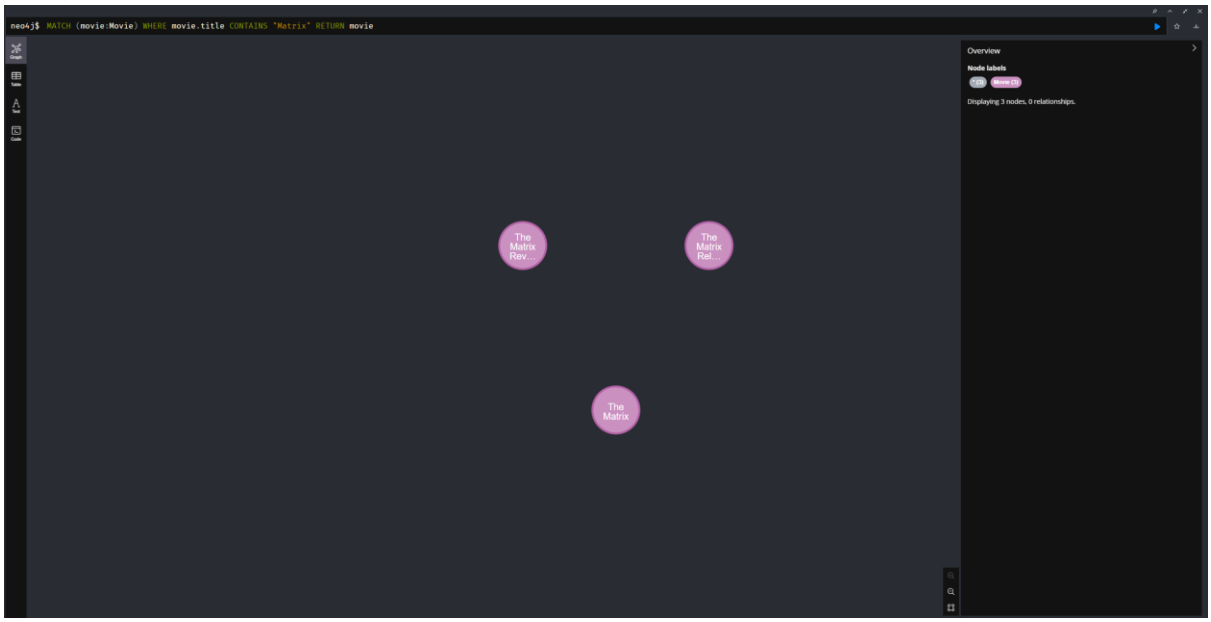
Functionality

This query allows us to search for movies based on their title. It utilizes the CONTAINS function to filter the results based on the provided title, returning all movies matching the specified search term.

Results

App

Neo4j



Query 3: Listing All Actors

- MATCH (person:Person) RETURN person

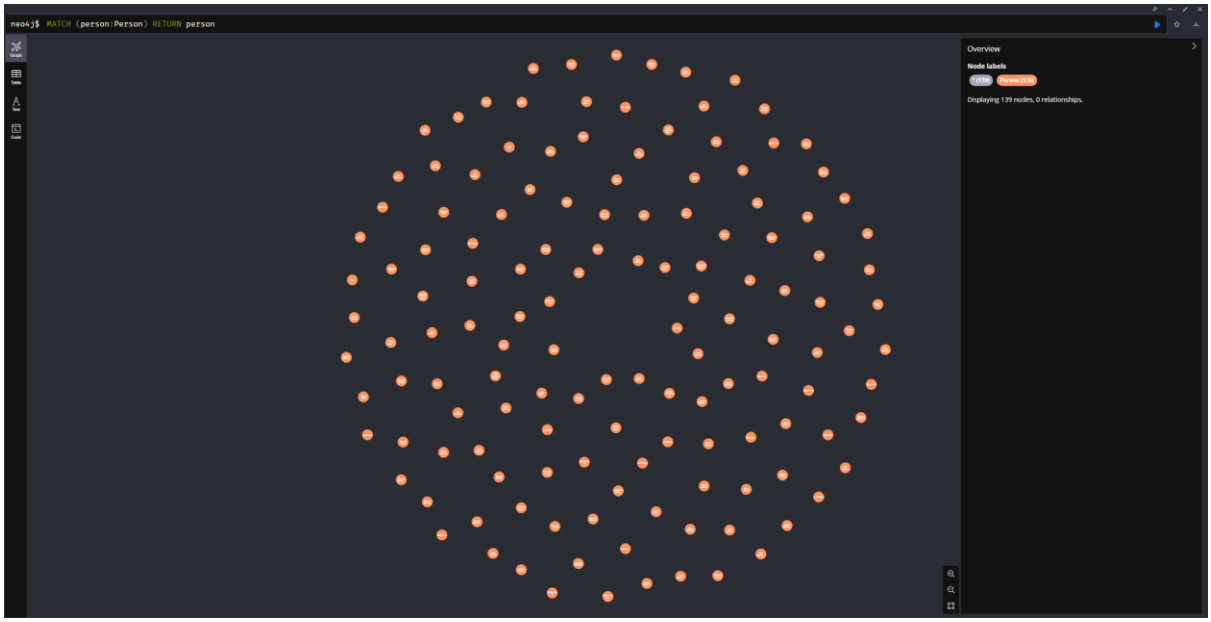
Functionality

This query retrieves a comprehensive list of all persons within the database. It utilizes the MATCH clause to traverse the entire graph, identifying and extracting all nodes representing actors.

Results

App

Neo4j



Query 4: Finding Movies Starring a Specific Person

- MATCH (person:Person)-[:ACTED_IN]->(movie:Movie) WHERE person.name CONTAINS $name RETURN movie

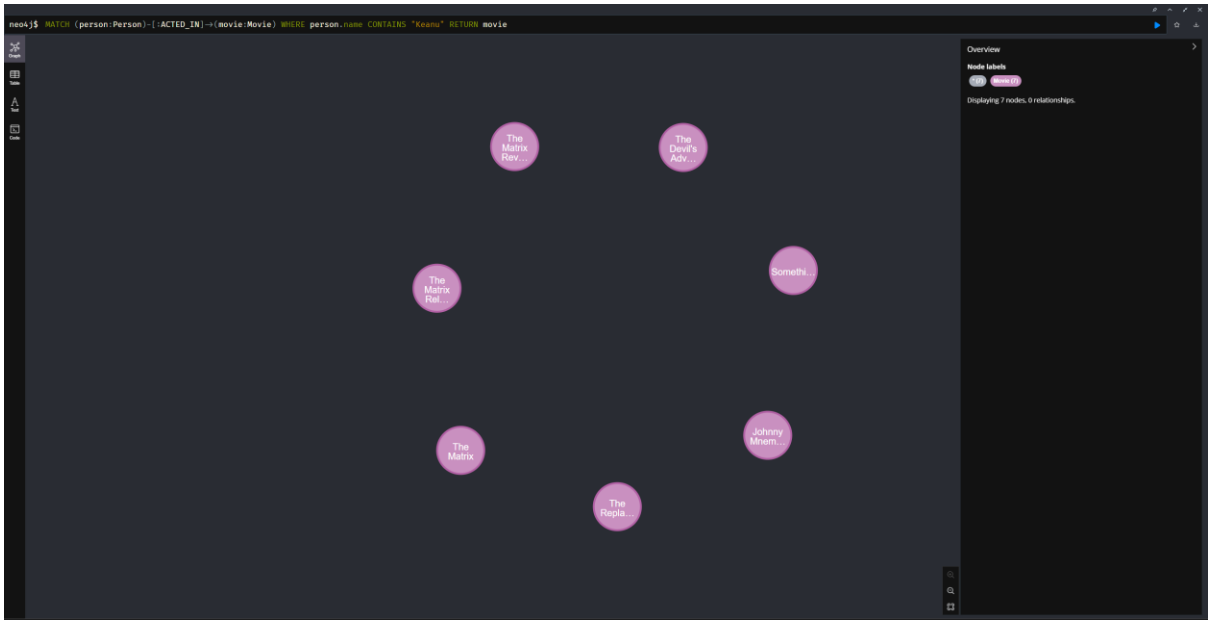
Functionality

This query retrieves a comprehensive list of all persons within the database. It utilizes the MATCH clause to traverse the entire graph, identifying and extracting all nodes representing actors.

Results

App

Neo4j



Constraints

To ensure data integrity, it has also defined the following constraints:

- CREATE CONSTRAINT uniqueName IF NOT EXISTS FOR (p:Person) REQUIRE (p.name) IS UNIQUE;
    - uniqueName: The p.name attribute of the Person class must be unique. This prevents duplicate names for people.
- CREATE CONSTRAINT uniqueTitle IF NOT EXISTS FOR (m:Movie) REQUIRE (m.title) IS UNIQUE;
    - uniqueTitle: The m.title attribute of the Movie class must be unique. This prevents duplicate titles for movies.

Constrains also have the advantage of indexing the attribute in question.

Indexes

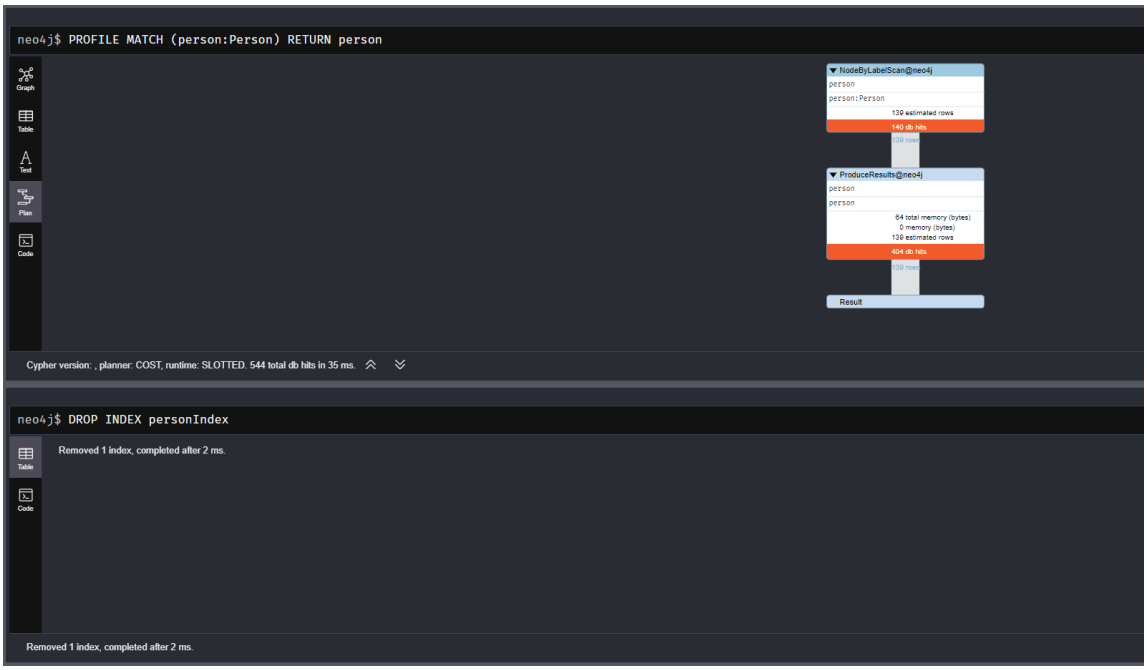
For improved query performance, we have also created indexes on the following attributes:

- CREATE INDEX personIndex IF NOT EXISTS FOR (p:Person) ON (p.name);
    - personIndex: The p.name attribute of the Person class. This index will speed up queries that involve searching for people by name.
- CREATE INDEX movieIndex IF NOT EXISTS FOR (m:Movie) ON (m.title, m.released);
    - movieIndex: The m.title and m.released attributes of the Movie class. This index will speed up queries that involve searching for movies by title or release date.
- CREATE INDEX personBornIndex IF NOT EXISTS FOR (p:Person) ON (p.born);
    - personBornIndex: The p.born attribute of the Person class. This index will speed up queries that involve searching for people by their birth date.
- CREATE INDEX movieReleasedIndex IF NOT EXISTS FOR (m:Movie) ON (m.released);
    - movieReleasedIndex: The m.released attribute of the Movie class. This index will speed up queries that involve searching for movies by their release date.

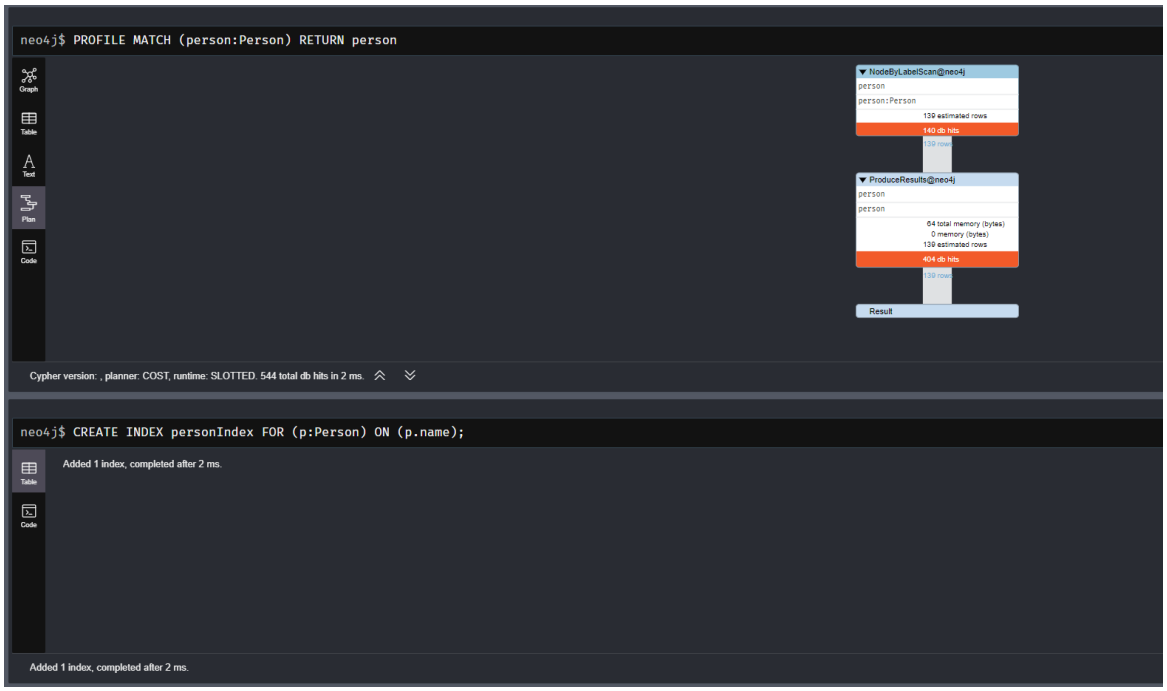
Profiling

In Neo4j, profiling is a crucial tool for evaluating the performance of queries and identifying potential bottlenecks. By using the PROFILE clause, you can gain insights into the execution of your queries, including the number of database hits, the runtime, and the query planner used.

In this example, we created an index on the p.name property of the Person class and executed the MATCH (person:Person) RETURN person query. Profiling revealed that there were 544 database hits in 2 ms, indicating efficient query execution.



We then dropped the index and re-executed the query. Profiling showed that there were still 544 database hits but with a slower execution time of 35 ms. This suggests that the index plays a significant role in optimizing query performance for searching by name.



Conclusion

Our project aimed to explore the use of Neo4j to create a graph database for managing and analyzing movie data. We successfully designed a data schema, developed Cypher queries, and employed profiling techniques to evaluate query performance.

Key Findings

- Neo4j's graph data model effectively captures the interconnectedness of movies, persons, and their relationships.
- Cypher queries provide a powerful and expressive language for navigating and extracting information from graph databases.
- Profiling is essential for identifying and optimizing query performance in Neo4j.

Challenges and Successes

- One challenge was understanding the intricacies of graph data modeling and querying. However, we overcame this by leveraging documentation and online resources.
- A key success was successfully implementing Cypher queries to extract and analyze movie data, demonstrating the power of Neo4j for graph-based data management.

Lessons Learned

- Thoroughly understanding the data model and relationships is crucial for effective query development.
- Profiling is essential for optimizing query performance and identifying potential performance bottlenecks.
- Neo4j's graph data model provides a flexible and powerful approach for managing and analyzing complex data relationships.