SOEN 363: Data Systems for Software Engineers

Section: S

Presented By:

Haris Mahmood - 40135271

Presented to:
Ali Jannatpour

Fall 2023

[40 pts] Write scripts to create the database in Neo4J:

1.
```
// Import Content Ratings
LOAD CSV WITH HEADERS FROM 'file:///ContentRating.csv' AS row
MERGE (cr:ContentRating {contentRating_id: toInteger(row.contentRating_id)})
SET cr.rating_name = row.rating_name;
```

2.
```
// Import Movies
LOAD CSV WITH HEADERS FROM 'file:///Movie.csv' AS row
MERGE (m:Movie {
  movie_id: toInteger(row.movie_id),
  imdb_id: row.imdb_id,
  title: row.title,
  plot: row.plot,
  rating: toFloat(row.rating),
  runtime: toInteger(row.runtime),
  num_reviews: toInteger(row.num_reviews),
  release_year: toInteger(row.release_year),
  akas: row.akas
})
// Connect Movie to Content Rating
MERGE (cr:ContentRating {contentRating_id: toInteger(row.content_rating_id)})
MERGE (m)-[:HAS_CONTENT_RATING]->(cr);
```

Note: Genre and language isn't part of Movie as an attribute and is its own separate node, on the basis of how my A2 was configured, this has been approved to be ok for this assignment by the TA Akshit Desai

3.
```
// Import Genres
LOAD CSV WITH HEADERS FROM 'file:///Genres.csv' AS row
MERGE (g:Genre {
  genres_id: toInteger(row.genres_id),
  genre_name: row.genre_name
})
// Connect Genre to Movie
MERGE (m:Movie {movie_id: toInteger(row.movie_id)})
MERGE (m)-[:HAS_GENRE]->(g);
```

4.

```
// Import Countries
LOAD CSV WITH HEADERS FROM 'file:///Country.csv' AS row
MERGE (c:Country {
  country_id: toInteger(row.country_id),
  country_name: row.country_name,
  country_code: row.country_code
})
// Connect Country to Movie
MERGE (m:Movie {movie_id: toInteger(row.movie_id)})
MERGE (m)-[:HAS_COUNTRY]->(c);
```

5.

```
// Import Movie Languages
LOAD CSV WITH HEADERS FROM 'file:///MovieLanguage.csv' AS row
MERGE (ml:MovieLanguage {
  language_id: toInteger(row.language_id),
  language_name: row.language_name
})
// Connect Language to Movie
MERGE (m:Movie {movie_id: toInteger(row.movie_id)})
MERGE (m)-[:HAS_LANGUAGE]->(ml);
```

6.

```
// Import Keywords
LOAD CSV WITH HEADERS FROM 'file:///Keywords.csv' AS row
MERGE (k:Keyword {
  keywords_id: toInteger(row.keywords_id),
  keyword_name: row.keyword_name
})
// Connect Keyword to Movie
MERGE (m:Movie {movie_id: toInteger(row.movie_id)})
MERGE (m)-[:HAS_KEYWORD]->(k);
```

7.

```
 // Import Persons
LOAD CSV WITH HEADERS FROM 'file:///Person.csv' AS row
MERGE (p:Person {
  person_id: toInteger(row.person_id),
  imdb_id: row.imdb_id,
  person_name: row.person_name
});
```

8.

```
// Import Actors with person_id and character_name
LOAD CSV WITH HEADERS FROM 'file:///Actor.csv' AS row
MERGE (a:Actor {actor_id: toInteger(row.actor_id)})
SET a.person_id = toInteger(row.person_id),
    a.character_name = row.character_name;


// Link Actors to Movies
LOAD CSV WITH HEADERS FROM 'file:///Actor.csv' AS row
MATCH (a:Actor {actor_id: toInteger(row.actor_id)})
MATCH (m:Movie {movie_id: toInteger(row.movie_id)})
MERGE (a)-[:ACTED_IN]->(m)

// Link Actors to Persons based on person_id
LOAD CSV WITH HEADERS FROM 'file:///Actor.csv' AS row
MATCH (a:Actor {actor_id: toInteger(row.actor_id)})
MATCH (p:Person {person_id: toInteger(row.person_id)})
MERGE (a)-[:HAS_PERSON]->(p);
```

9.

```
// Import Directors with person_id
LOAD CSV WITH HEADERS FROM 'file:///Director.csv' AS row
MERGE (d:Director {director_id: toInteger(row.director_id)})
SET d.person_id = toInteger(row.person_id);

// Link Directors to Movies based on movie_id
LOAD CSV WITH HEADERS FROM 'file:///Director.csv' AS row
MATCH (d:Director {director_id: toInteger(row.director_id)})
MATCH (m:Movie {movie_id: toInteger(row.movie_id)})
MERGE (d)-[:DIRECTED]->(m);

// Link Directors to Persons based on person_id
LOAD CSV WITH HEADERS FROM 'file:///Director.csv' AS row
MATCH (d:Director {director_id: toInteger(row.director_id)})
MATCH (p:Person {person_id: toInteger(row.person_id)})
MERGE (d)-[:IS_PERSON]->(p);
```

10.

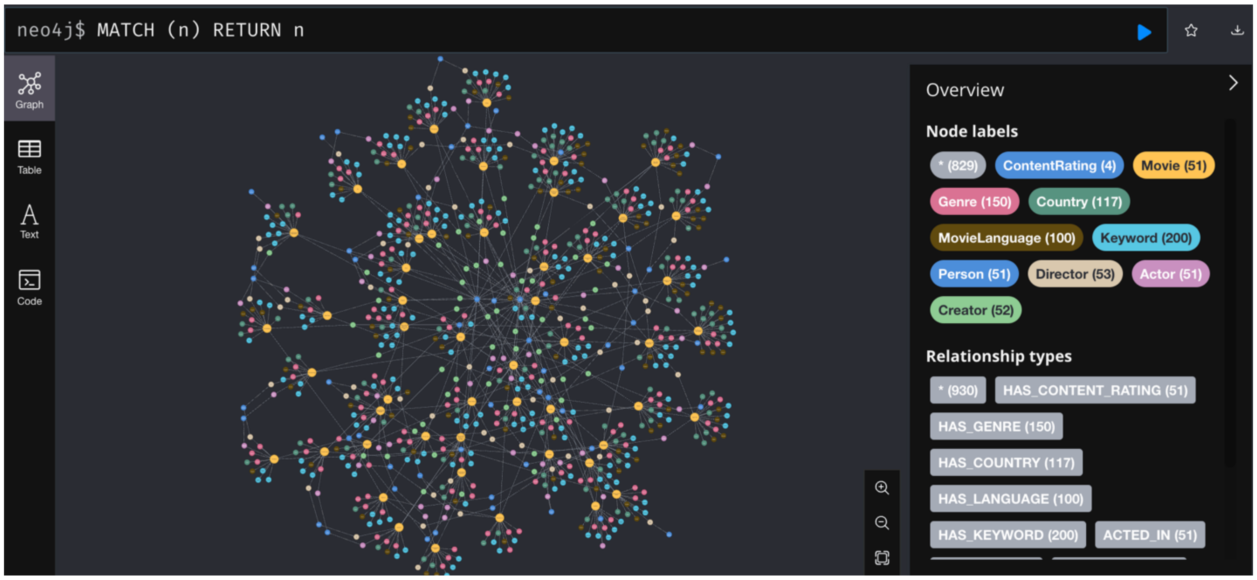
```
// Import Creators with person_id and link to Movies
LOAD CSV WITH HEADERS FROM 'file:///Creator.csv' AS row
MERGE (c:Creator {creator_id: toInteger(row.creator_id)})
SET c.person_id = toInteger(row.person_id);

// Link Creators to Persons based on person_id
LOAD CSV WITH HEADERS FROM 'file:///Creator.csv' AS row
MATCH (c:Creator {creator_id: toInteger(row.creator_id)})
MATCH (p:Person {person_id: toInteger(row.person_id)})
MERGE (c)-[:IS_PERSON]->(p);

// Link Creators to Movies based on movie_id
LOAD CSV WITH HEADERS FROM 'file:///Creator.csv' AS row
MATCH (c:Creator {creator_id: toInteger(row.creator_id)})
MATCH (m:Movie {movie_id: toInteger(row.movie_id)})
MERGE (c)-[:CREATED]->(m);
```

Queries:

All the nodes and relationships:



```
neo4j$ MATCH (n) RETURN n
```

Overview

Node labels

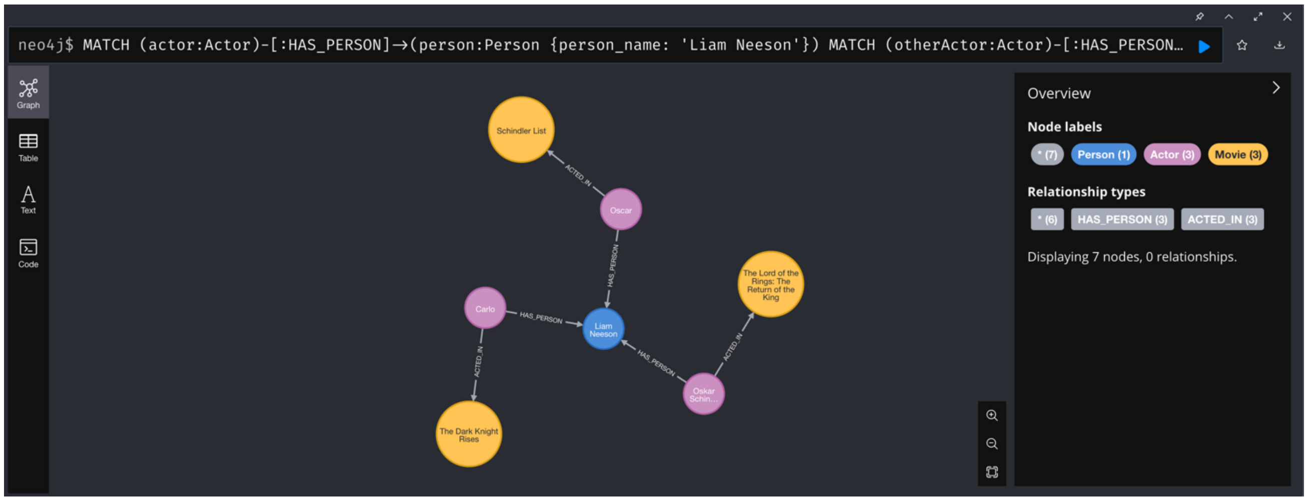
* (829) ContentRating (4) Movie (51) Genre (150) Country (117) MovieLanguage (100) Keyword (200) Person (51) Director (53) Actor (51) Creator (52)

Relationship types

* (930) HAS_CONTENT_RATING (51) HAS_GENRE (150) HAS_COUNTRY (117) HAS_LANGUAGE (100) HAS_KEYWORD (200) ACTED_IN (51)

A) **[5 pts] Find all movies that are played by a sample actor.**

Ans:
MATCH (actor:Actor)-[:HAS_PERSON]->(person:Person {person_name: 'Liam Neeson'})
MATCH (otherActor:Actor)-[:HAS_PERSON]->(person)
MATCH (otherActor)-[:ACTED_IN]->(movie:Movie)
RETURN person, otherActor, movie;



```
neo4j$ MATCH (actor:Actor)-[:HAS_PERSON]→(person:Person {person_name: 'Liam Neeson'}) MATCH (otherActor:Actor)-[:HAS_PERSON…
```

Overview

Node labels

* (7) Person (1) Actor (3) Movie (3)

Relationship types

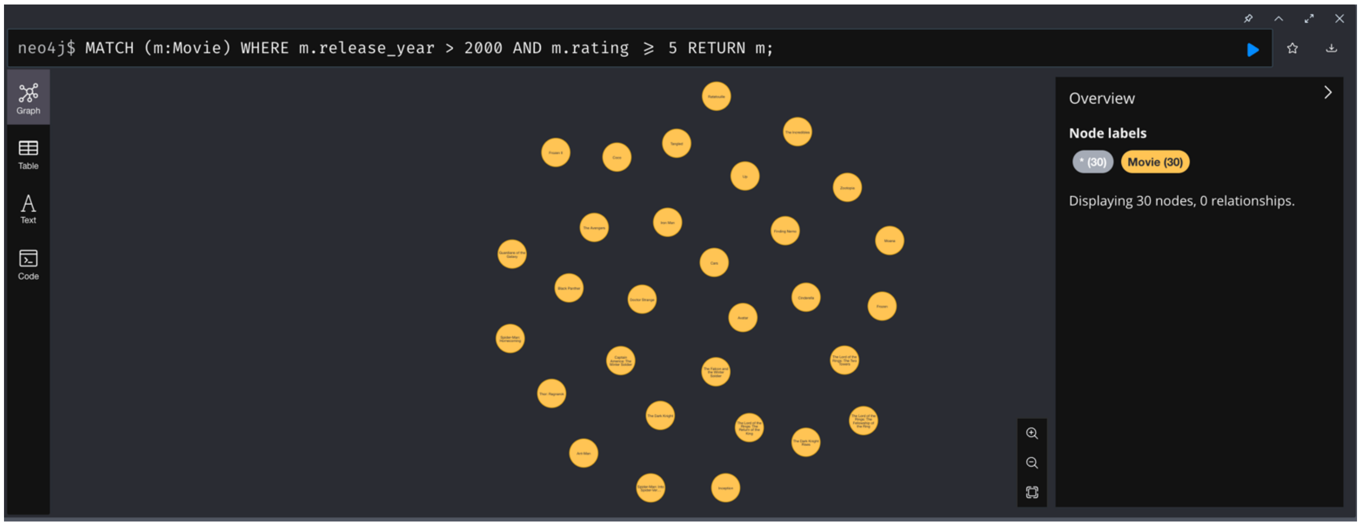
* (6) HAS_PERSON (3) ACTED_IN (3)

Displaying 7 nodes, 0 relationships.

B) [5 pts] Find all movies that are released after the year 2000 and has a rating of at least 5.

Ans:

```
MATCH (m:Movie)
WHERE m.release_year > 2000 AND m.rating >= 5
RETURN m;
```



```
neo4j$ MATCH (m:Movie) WHERE m.release_year > 2000 AND m.rating ≥ 5 RETURN m;
```

Overview

Node labels

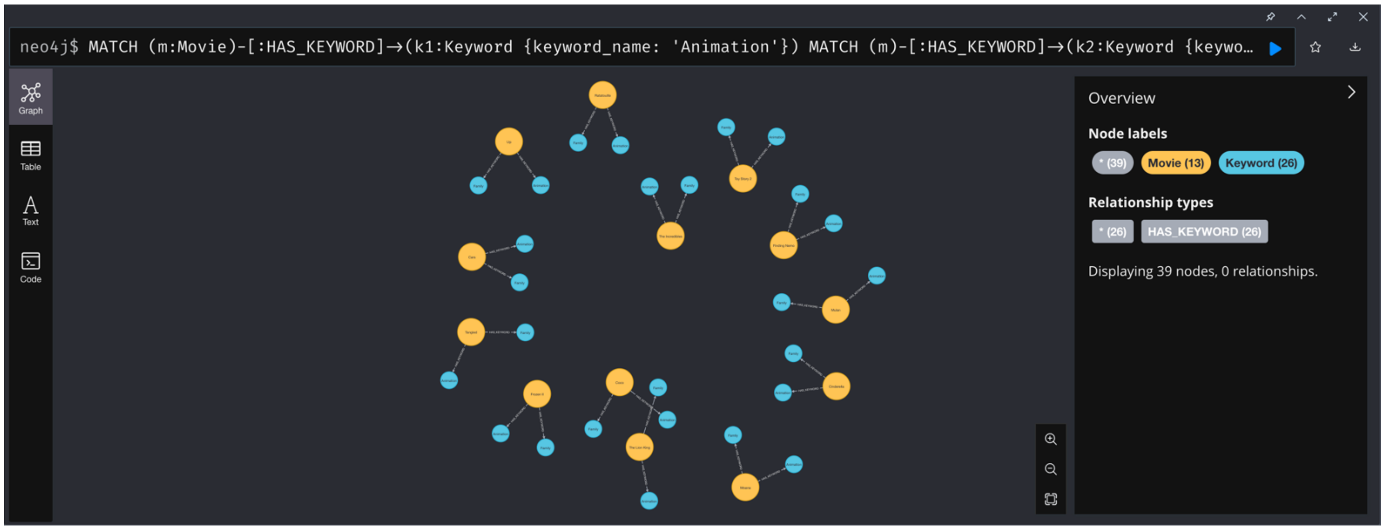
* (30) Movie (30)

Displaying 30 nodes, 0 relationships.

C) Find all movies that share two keywords of your choice. Make sure your query returns more than one movie.

Ans:

```
MATCH (m:Movie)-[:HAS_KEYWORD]->(k1:Keyword {keyword_name:
'Animation'})
MATCH (m)-[:HAS_KEYWORD]->(k2:Keyword {keyword_name: 'Family'})
RETURN m, k1, k2;
```



```
neo4j$ MATCH (m:Movie)-[:HAS_KEYWORD]→(k1:Keyword {keyword_name: 'Animation'}) MATCH (m)-[:HAS_KEYWORD]→(k2:Keyword {keywo…
```

Overview

Node labels

* (39) Movie (13) Keyword (26)

Relationship types

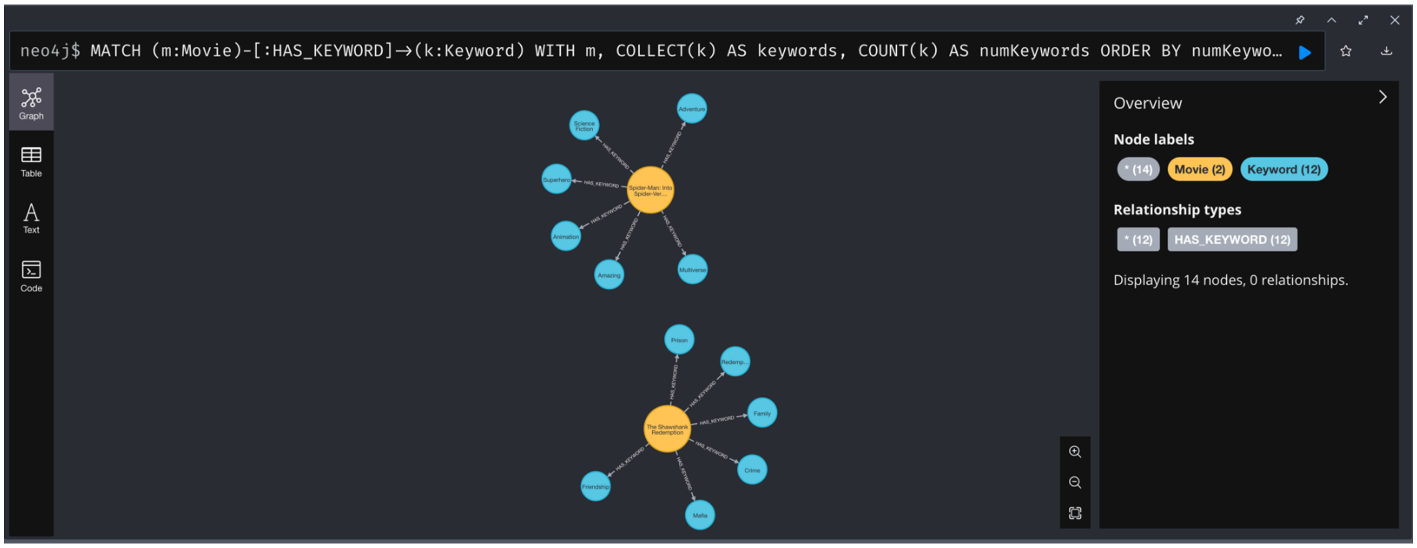
* (26) HAS_KEYWORD (26)

Displaying 39 nodes, 0 relationships.

D) [10 pts] Find top 2 movies with largest number of keywords.

Ans:

```
MATCH (m:Movie)-[:HAS_KEYWORD]->(k:Keyword)
WITH m, COLLECT(k) AS keywords, COUNT(k) AS numKeywords
ORDER BY numKeywords DESC
LIMIT 2
RETURN m, numKeywords, keywords;
```



```
neo4j$ MATCH (m:Movie)-[:HAS_KEYWORD]→(k:Keyword) WITH m, COLLECT(k) AS keywords, COUNT(k) AS numKeywords ORDER BY numKeywo…
```

Overview

Node labels

* (14) Movie (2) Keyword (12)

Relationship types

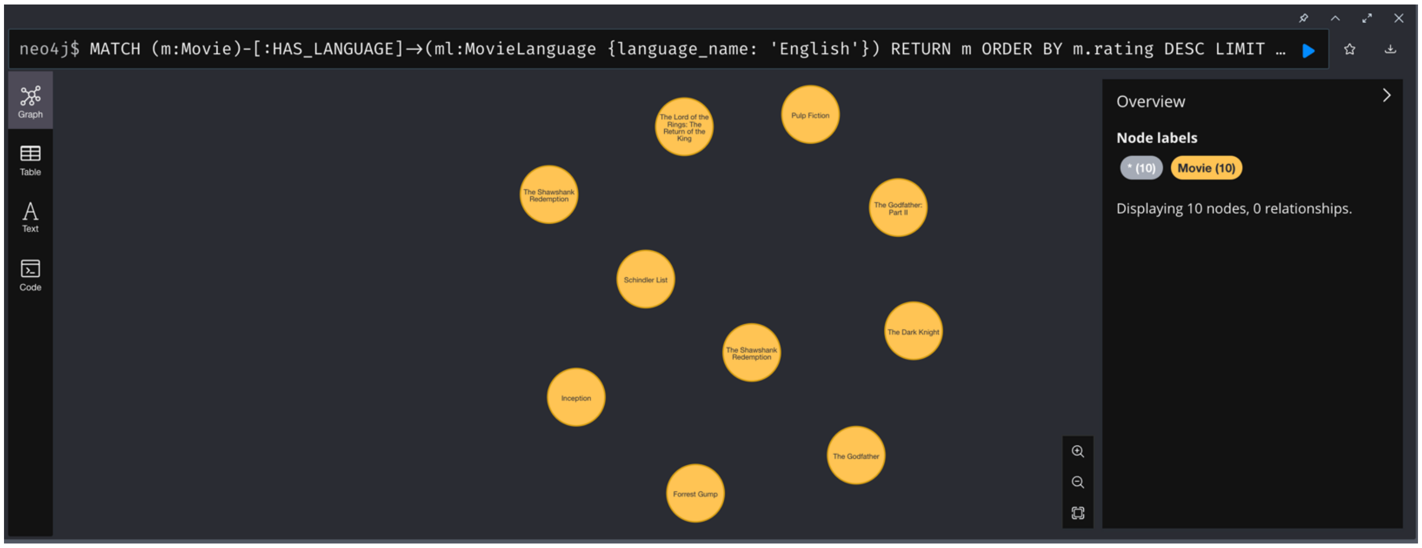
* (12) HAS_KEYWORD (12)

Displaying 14 nodes, 0 relationships.

E) [10 pts] Find top 10 movies (ordered by rating) in a language of your choice.

Ans:

```
MATCH (m:Movie)-[:HAS_LANGUAGE]->(ml:MovieLanguage {language_name:
'English'})
RETURN m
ORDER BY m.rating DESC
LIMIT 10;
```



```
neo4j$ MATCH (m:Movie)-[:HAS_LANGUAGE]→(ml:MovieLanguage {language_name: 'English'}) RETURN m ORDER BY m.rating DESC LIMIT …
```

Overview

Node labels

* (10) Movie (10)

Displaying 10 nodes, 0 relationships.

F) [5 pts] Build full text search index to query movie plots.

Ans:

CALL db.index.fulltext.createNodeIndex("plotIndex", ["Movie"], ["plot"]);

G) [5 pts] Write a full text search query and search for some sample text of your choice.

Ans:

CALL db.index.fulltext.queryNodes("plotIndex", " 'Two imprisoned men bond over a number of years, finding solace and eventual redemption through acts of common decency.'")
YIELD node, score
RETURN node, score;