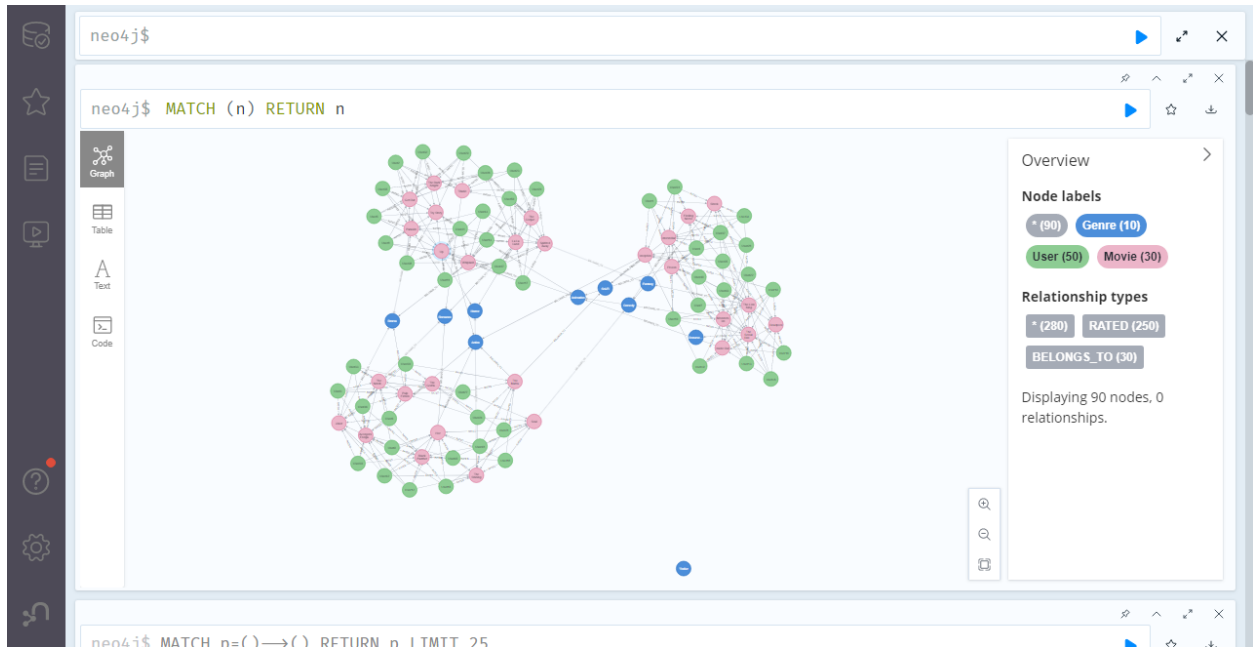


Kaziah Ofori

This graph database models the interactions between users, films, and genres in order to simulate a movie recommendation system. For analytics and query testing, this graph aims to replicate the composition and actions of an actual recommendation system. Using actual data from the MovieLens 100K dataset, which was made available by GroupLens Research, this graph database simulates a movie recommendation engine. To facilitate effective relationship inquiries, recommendation algorithms, and user behavior analysis, the data has been organized into a graph format. There are 100,000 user ratings for 1,682 films in the dataset. Every rating has a timestamp, user ID, movie ID, and rating (1–5). Genres and titles are included in movie metadata. In my graph I had three different node types, The user, which had a sample subset of 50 users and properties based on the users ID, age, gender, and occupation. The second node type was movie, which had the properties movie id, title. and year and had a subset of 30-50 movies. The third node consisted of subsets derived from the genre flags in the dataset. Its only had one property called name, for example drama, comedy, romance. For the relationships, I first made a related relationship from user to movie which represents the user's rating of the movie. Then, belongs to relationships type from movie to genre to show the genres each movie falls under.

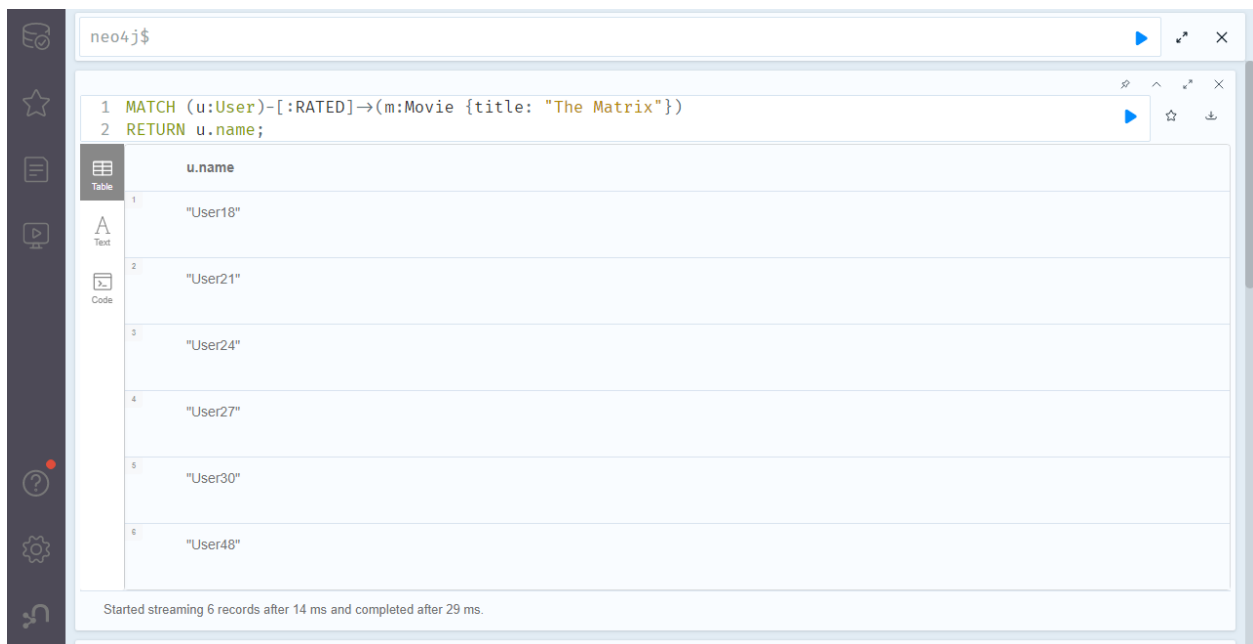
Link to dataset: [Movie Lens](#)

Match and Return



1. Find all users that rated "The Matrix"

```
MATCH (u:User)-[:RATED]->(m:Movie {title: "The Matrix"})
RETURN u.name;
```



2. List all movies rated higher than 4.

```
MATCH (u:User)-[r:RATED]->(m:Movie)
```

```
WHERE r.rating > 4
```

```
RETURN m.title, r.rating, u.name;
```

The image shows the Neo4j Desktop interface. At the top, a command bar contains the prompt 'neo4j\$'. Below it, a query editor shows the following Cypher query: `neo4j$ MATCH (:User)-[r:RATED]->(m:Movie) RETURN m.title AS movie, COUNT(r) AS numRatings ORDER BY nu...`. The results are displayed in a table view with two columns: 'movie' and 'numRatings'. The table contains five rows of data, all with a rating of 10. A status bar at the bottom of the table indicates: 'Started streaming 5 records after 8 ms and completed after 10 ms.' Below the table, another query editor shows a different query: `neo4j$ MATCH (:User)-[r:RATED]->(m:Movie) RETURN m.title AS movie, r.timestamp AS ratedAt ORDER BY ra...`.

	movie	numRatings
1	"Avengers: Endgame"	10
2	"The Lion King"	10
3	"Frozen"	10
4	"Parasite"	10
5	"Up"	10

Started streaming 5 records after 8 ms and completed after 10 ms.

3. List all genres for the movie Interception.

```
MATCH (m:Movie {title: "Interception"})-[:BELONGS_TO]->(g:Genre)
```

```
RETURN g.name;
```

neo4j\$

```
1 MATCH (m:Movie {title: "Inception"})-[:BELONGS_TO]->(g:Genre)
2 RETURN g.name;
```

	g.name
1	"Action"
2	"Sci-Fi"

Started streaming 2 records after 10 ms and completed after 12 ms.

neo4j\$ MATCH (u:User)-[r:RATED]->(m:Movie) WHERE r.rating > 4 RETURN m.title, r.rating, u.name;

	m.title	r.rating	u.name
1	"The Matrix"	5.0	"User18"

4. Find the top 5 movies rated by user 48.

```
MATCH (:User {name: "User48"})-[r:RATED]->(m:Movie)
```

```
RETURN m.title, r.rating
```

```
ORDER BY r.rating DESC
```

```
LIMIT 5;
```

neo4j\$

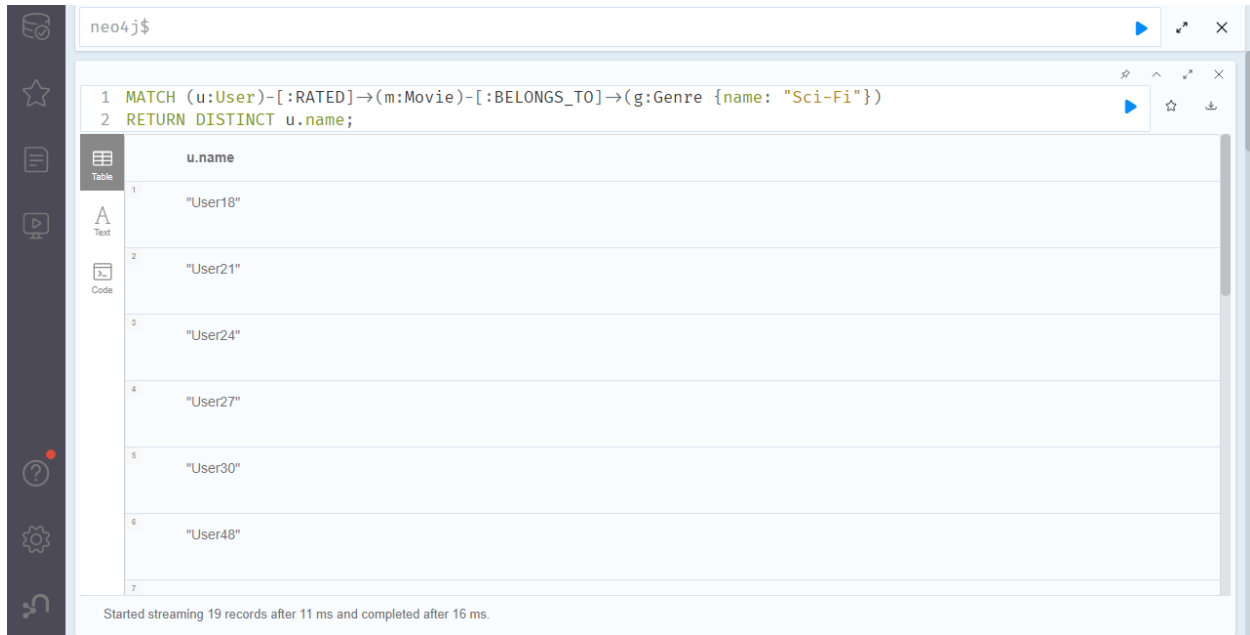
```
1 MATCH (:User {name: "User48"})-[r:RATED]->(m:Movie)
2
3 RETURN m.title, r.rating
4
5 ORDER BY r.rating DESC
6
7 LIMIT 5;
```

	m.title	r.rating
1	"The Matrix"	5.0
2	"The Shining"	4.0
3	"Black Panther"	3.0
4	"Soul"	2.0
5	"Her"	1.0

Started streaming 5 records after 10 ms and completed after 11 ms.

5. Find people who like the genre Sci-Fi.

```
MATCH (u:User)-[:RATED]->(m:Movie)-[:BELONGS_TO]->(g:Genre {name: "Sci-Fi"})
RETURN DISTINCT u.name;
```



The screenshot shows the Neo4j browser interface. At the top, the address bar displays 'neo4j\$'. Below it, the Cypher query editor contains two lines of code: `1 MATCH (u:User)-[:RATED]->(m:Movie)-[:BELONGS_TO]->(g:Genre {name: "Sci-Fi"})` and `2 RETURN DISTINCT u.name;`. The results are displayed in a table view with a single column labeled 'u.name'. The table contains six rows of user names: 'User18', 'User21', 'User24', 'User27', 'User30', and 'User48'. A status bar at the bottom indicates 'Started streaming 19 records after 11 ms and completed after 16 ms.'

	u.name
1	"User18"
2	"User21"
3	"User24"
4	"User27"
5	"User30"
6	"User48"
7	

6. Find the genres most associated with high ratings.

```
MATCH (u:User)-[r:RATED]->(m:Movie)-[:BELONGS_TO]->(g:Genre)
WHERE r.rating > 4.0
RETURN g.name, COUNT(*) AS freq
ORDER BY freq DESC;
```

neo4j\$

```

1 MATCH (u:User)-[r:RATED]-(m:Movie)-[:BELONGS_TO]-(g:Genre)
2 WHERE r.rating > 4.0
3 RETURN g.name, COUNT(*) AS freq
4 ORDER BY freq DESC;

```

	g.name	freq
1	"Animation"	9
2	"Action"	8
3	"Drama"	7
4	"Romance"	7
5	"Comedy"	5
6	"Sci-Fi"	4
7		

7. Find average rating per movie.

```
MATCH (u:User)-[r:RATED]->(m:Movie)
```

```
RETURN m.title, AVG(r.rating) AS avg_rating;
```

neo4j\$

```

neo4j$ MATCH (u:User)-[r:RATED]-(m:Movie) RETURN m.title, AVG(r.rating) AS avg_rating;

```

	m.title	avg_rating
1	"The Matrix"	3.3333333333333335
2	"Inception"	3.0
3	"Titanic"	2.857142857142857
4	"The Godfather"	3.0
5	"Shrek"	3.142857142857143
6	"The Dark Knight"	3.0
7		

Started streaming 30 records after 8 ms and completed after 15 ms.

8. List movies that belong to multiple genres

```
MATCH (m:Movie)-[:BELONGS_TO]->(g:Genre)
```

```
WITH m, COUNT(g) AS genre_count
```

```
WHERE genre_count > 1
```

```
RETURN m.title, genre_count;
```

neo4j\$

```

1 MATCH (m:Movie)-[:BELONGS_TO]->(g:Genre)
2 WITH m, COUNT(g) AS genre_count
3 WHERE genre_count > 1
4 RETURN m.title, genre_count;

```

	m.title	genre_count
1	"The Matrix"	2
2	"Inception"	2

Started streaming 2 records after 10 ms and completed after 14 ms.

neo4j\$ MATCH (:User {name: "User48"})-[:RATED]->(m:Movie) RETURN m.title, r.rating ORDER BY r.rating...

	m.title	r.rating
1	"The Matrix"	5.0

9. Find movies similar by genre

```

MATCH (m1:Movie {title: "The Matrix"})-[:BELONGS_TO]->(g:Genre)-[:BELONGS_TO]-(m2:Movie)
WHERE m1 <> m2
RETURN DISTINCT m2.title;

```

neo4j\$

```

1 MATCH (m1:Movie {title: "The Matrix"})-[:BELONGS_TO]->(g:Genre)-[:BELONGS_TO]-(m2:Movie)
2 WHERE m1 <> m2
3 RETURN DISTINCT m2.title;

```

	m2.title
1	"Inception"
2	"The Dark Knight"
3	"Avengers: Endgame"
4	"Black Panther"
5	"Interstellar"

Started streaming 5 records after 9 ms and completed after 11 ms.

neo4j\$ MATCH (m:Movie)-[:BELONGS_TO]->(g:Genre) WITH m, COUNT(g) AS genre_count WHERE genre_count > 1...

10. Find average ratings by each genre.

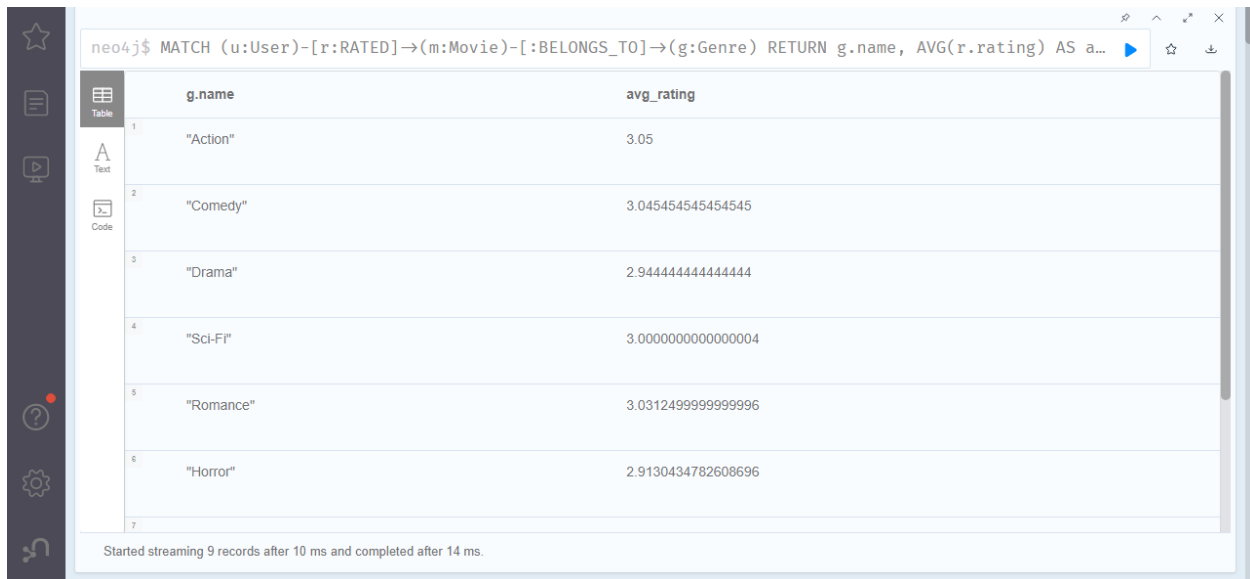
```

MATCH (u:User)-[:RATED]->(m:Movie)-[:BELONGS_TO]->(g:Genre)

```



```
RETURN g.name, AVG(r.rating) AS avg_rating;
```



The screenshot shows a Neo4j Cypher query interface. The query entered is: `neo4j$ MATCH (u:User)-[r:RATED]->(m:Movie)-[:BELONGS_TO]->(g:Genre) RETURN g.name, AVG(r.rating) AS a...`. The results are displayed in a table with two columns: `g.name` and `avg_rating`. The table contains 6 rows of data. A status bar at the bottom indicates: "Started streaming 9 records after 10 ms and completed after 14 ms."

	g.name	avg_rating
1	"Action"	3.05
2	"Comedy"	3.045454545454545
3	"Drama"	2.944444444444444
4	"Sci-Fi"	3.0000000000000004
5	"Romance"	3.0312499999999996
6	"Horror"	2.9130434782608696
7		

11. Find most active users by rating count

```
MATCH (u:User)-[r:RATED]->()
```

```
RETURN u.name, COUNT(r) AS ratings
```

```
ORDER BY ratings DESC
```

```
LIMIT 5;
```

neo4j\$

```

1 MATCH (u:User)-[r:RATED]->()
2 RETURN u.name, COUNT(r) AS ratings
3 ORDER BY ratings DESC
4 LIMIT 5;

```

	u.name	ratings
1	"User2"	5
2	"User3"	5
3	"User4"	5
4	"User5"	5
5	"User1"	5

Started streaming 5 records after 9 ms and completed after 12 ms.

12. Find the rating distribution for The Matrix

MATCH (:User)-[r:RATED]->(m:Movie {title: "The Matrix"})

RETURN r.rating, COUNT(*) AS freq

ORDER BY r.rating;

neo4j\$

```

1 MATCH (:User)-[r:RATED]->(m:Movie {title: "The Matrix"})
2 RETURN r.rating, COUNT(*) AS freq
3 ORDER BY r.rating;

```

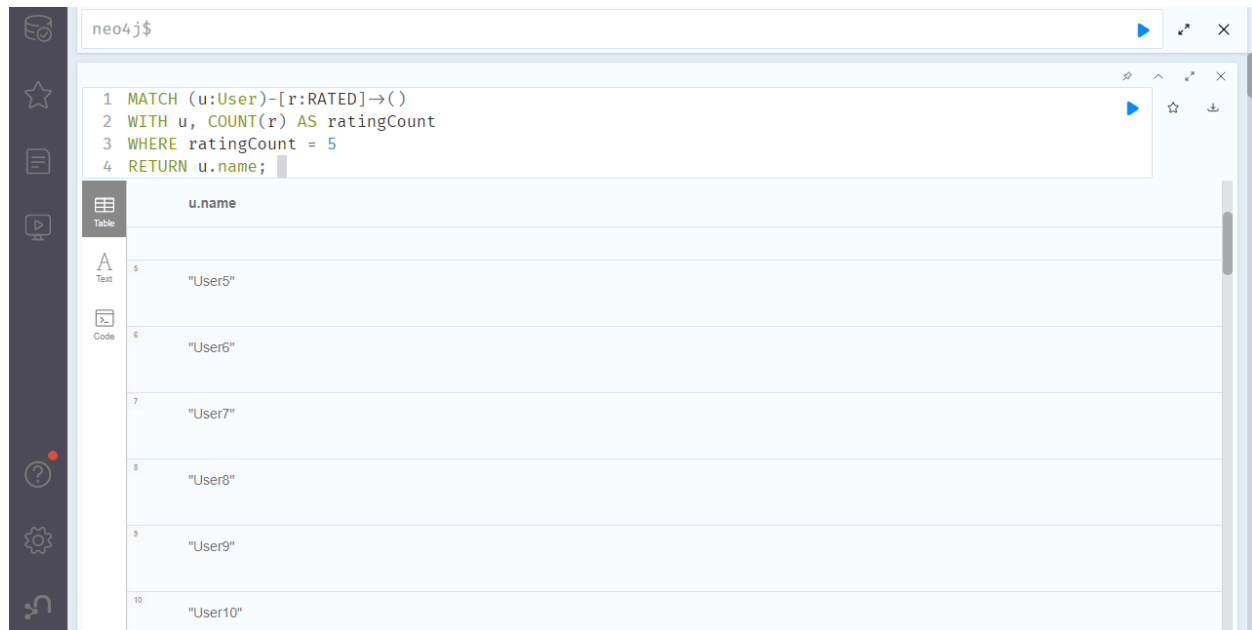
	r.rating	freq
1	1.0	1
2	2.0	1
3	3.0	1
4	4.0	1
5	5.0	2

Started streaming 5 records after 7 ms and completed after 8 ms.

neo4j\$ MATCH (u:User)-[r:RATED]->() RETURN u.name, COUNT(r) AS ratings ORDER BY ratings DESC LIMIT 5;

13. Find users that have rated 5 movies.

```
MATCH (u:User)-[r:RATED]->()
WITH u, COUNT(r) AS ratingCount
WHERE ratingCount = 5
RETURN u.name;
```



The screenshot shows the Neo4j Cypher Shell interface. At the top, the address bar displays 'neo4j\$'. Below it, a query editor contains the following Cypher query:

```
1 MATCH (u:User)-[r:RATED]->()
2 WITH u, COUNT(r) AS ratingCount
3 WHERE ratingCount = 5
4 RETURN u.name;
```

Below the query editor, the results are displayed in a table format. The table has a single column labeled 'u.name'. The results are as follows:

	u.name
5	"User5"
6	"User6"
7	"User7"
8	"User8"
9	"User9"
10	"User10"

14. Find the most common genre.

```
MATCH (:Movie)-[:BELONGS_TO]->(g:Genre)
RETURN g.name, COUNT(*) AS movie_count
ORDER BY movie_count DESC
LIMIT 1;
```

The image shows the Neo4j Desktop interface with two queries executed. The first query counts the number of movies in the 'Action' genre. The second query finds the user 'User5' based on a rating of 5.

```
neo4j$ MATCH (:Movie)-[:BELONGS_TO]-(g:Genre) RETURN g.name, COUNT(*) AS movie_count ORDER BY movie_...
```

g.name	movie_count
"Action"	5

Started streaming 1 records after 8 ms and completed after 9 ms.

```
neo4j$ MATCH (u:User)-[:RATED]->() WITH u, COUNT(r) AS ratingCount WHERE ratingCount = 5 RETURN u.name...
```

u.name
"User5"

15.Delete a rating

```
MATCH (u:User {name: "Bob"})-[:RATED]->(m:Movie {title: "Titanic"})
DELETE r
```

The image shows the Neo4j Desktop interface. The first query deletes a rating for 'User2' on the movie 'Titanic'. The second query is the same as the one in the previous image, showing the 'Action' genre has 5 movies.

```
1 MATCH (u:User {name: "User2"})-[:RATED]->(m:Movie {title: "Titanic"})
2 DELETE r
```

Deleted 1 relationship, completed after 35 ms.

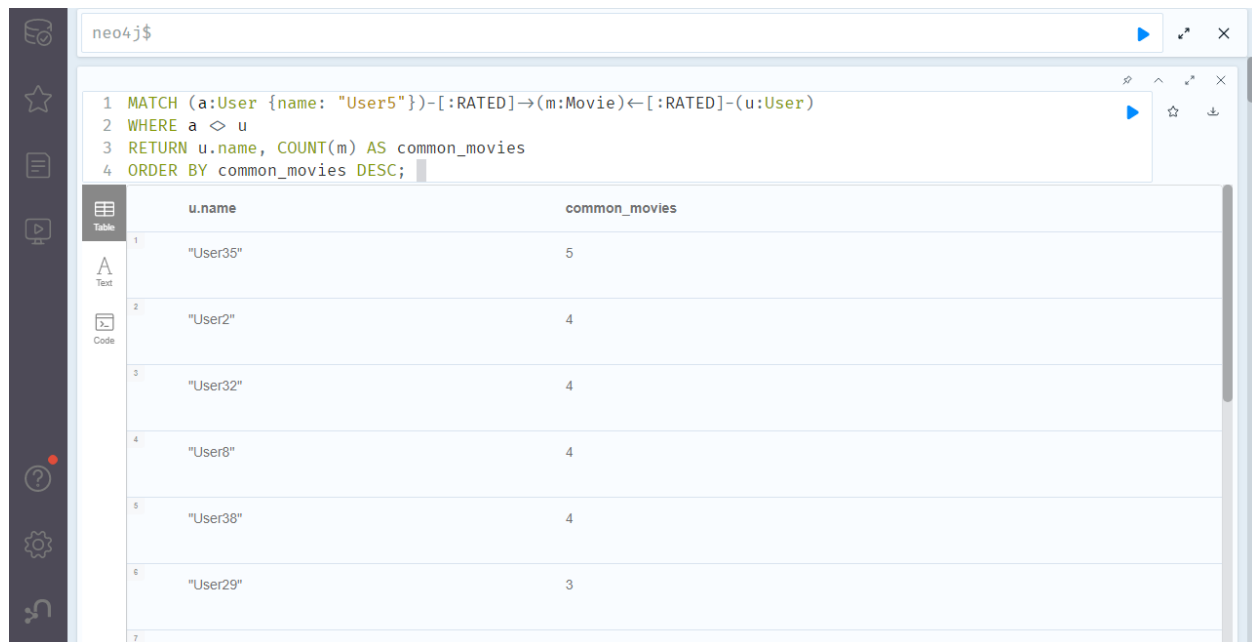
```
neo4j$ MATCH (:Movie)-[:BELONGS_TO]-(g:Genre) RETURN g.name, COUNT(*) AS movie_count ORDER BY movie_...
```

g.name	movie_count
"Action"	5

16.Find users who have rated similar movies to User 5.

```
MATCH (a:User {name: "User5"})-[:RATED]->(m:Movie)-[:RATED]-(u:User)
WHERE a <> u
RETURN u.name, COUNT(m) AS common_movies
```

ORDER BY common_movies DESC;



The screenshot shows the Neo4j browser interface. At the top, the address bar displays 'neo4j\$'. Below it, a code editor contains a Cypher query:

```
1 MATCH (a:User {name: "User5"})-[:RATED]->(m:Movie)-[:RATED]-(u:User)
2 WHERE a <> u
3 RETURN u.name, COUNT(m) AS common_movies
4 ORDER BY common_movies DESC;
```

Below the query editor, the results are displayed in a table view. The table has two columns: 'u.name' and 'common_movies'. The results are sorted in descending order of 'common_movies'.

	u.name	common_movies
1	"User35"	5
2	"User2"	4
3	"User32"	4
4	"User8"	4
5	"User38"	4
6	"User29"	3
7		

17. Find users who have only rated action movies.

```
MATCH (u:User)-[r:RATED]->(m:Movie)
```

```
WHERE NOT EXISTS {
```

```
  MATCH (m)-[:BELONGS_TO]->(g:Genre)
```

```
  WHERE g.name <> "Action"
```

```
}
```

```
RETURN DISTINCT u.name;
```



18. Find users who have rated only rated romance movies.

```
MATCH (u:User)-[r:RATED]->(m:Movie)
WHERE NOT EXISTS {
  MATCH (m)-[:BELONGS_TO]->(g:Genre)
  WHERE g.name <> "Romance"
}
RETURN DISTINCT u.name;
```

neo4j\$

```

1 MATCH (u:User)-[r:RATED]→(m:Movie)
2 WHERE NOT EXISTS {
3   MATCH (m)-[:BELONGS_TO]→(g:Genre)
4   WHERE g.name <> "Romance"
5 }
6 RETURN DISTINCT u.name;

```

	u.name
1	"User20"
2	"User23"
3	"User26"
4	"User29"
5	"User32"
6	"User50"

19. Find the most recently rated movie.

MATCH (:User)-[r:RATED]->(m:Movie)

RETURN m.title AS movie, r.timestamp AS ratedAt

ORDER BY ratedAt DESC

LIMIT 5;

neo4j\$

```

neo4j$ MATCH (:User)-[r:RATED]→(m:Movie) RETURN m.title AS movie, r.timestamp AS ratedAt ORDER BY ra...

```

	movie	ratedAt
1	"Inception"	1746996439348
2	"Shrek"	1746996409348
3	"The Godfather"	1746996399348
4	"The Dark Knight"	1746996389348
5	"Shrek"	1746996379348

Started streaming 5 records after 7 ms and completed after 9 ms.

```

neo4j$ MATCH (u:User)-[r:RATED]→(m:Movie) WHERE NOT EXISTS { MATCH (m)-[:BELONGS_TO]→(g:Genre) WHER...

```

20. Find the top 5 most rated movies.

```
MATCH (:User)-[r:RATED]->(m:Movie)
```

```
RETURN m.title AS movie, COUNT(r) AS numRatings
```

```
ORDER BY numRatings DESC
```

```
LIMIT 5;
```

The image shows the Neo4j Desktop interface. At the top, a command bar contains the prompt 'neo4j\$'. Below it, a query editor displays the following Cypher query:

```
neo4j$ MATCH (:User)-[r:RATED]->(m:Movie) RETURN m.title AS movie, COUNT(r) AS numRatings ORDER BY numRatings DESC LIMIT 5;
```

The results are displayed in a table view with two columns: 'movie' and 'numRatings'. The table contains five rows of data:

	movie	numRatings
1	"Avengers: Endgame"	10
2	"The Lion King"	10
3	"Frozen"	10
4	"Parasite"	10
5	"Up"	10

Below the table, a status message reads: 'Started streaming 5 records after 8 ms and completed after 10 ms.' At the bottom, another command bar shows a partial query: 'neo4j\$ MATCH (:User)-[r:RATED]->(m:Movie) RETURN m.title AS movie, r.timestamp AS ratedAt ORDER BY ra...'.