GRAPH THEORY AND ITS APPLICATIONS – ASSIGNMENT 4

Bollywood using neo4j

NOVEMBER 16, 2022

PES UNIVERSITY

The dataset is from https://boxofficeindia.com

A.Prashanth	**PES1UG20CS525**	**Sec: I**
Prateek Keshri	**PES1UG20CS526**	**Sec: I**

1. Prepare and import the data

```
LOAD CSV WITH HEADERS FROM 'file:///movie.csv'
AS row MERGE (m:movie {name: row.name, release_period:
row.release_period, remake: row.remake, franchise:
row.franchise, genre:row.genre, screens:
toInteger(row.screens), revenue:
toInteger(row.revenue),
budget:toInteger(row.budget)});

CREATE CONSTRAINT ON (m:movie) ASSERT m.name IS
UNIQUE;
```

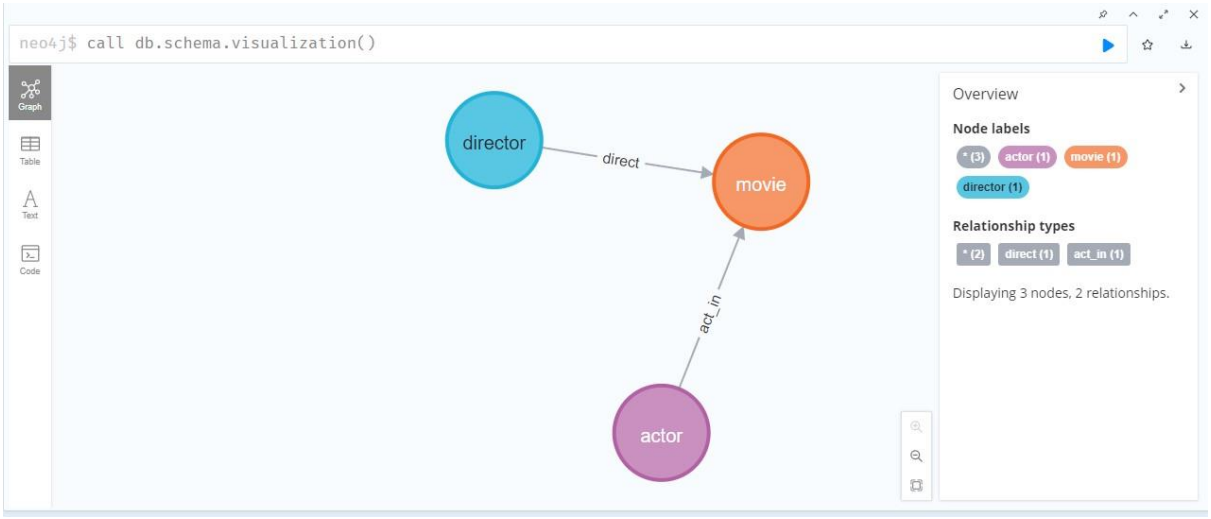
```
LOAD CSV WITH HEADERS FROM
'file:///director_movie.csv'
AS row MERGE (d:director {name: row.director}) MERGE
(m:movie {name: row.movie}) MERGE (d)-[r:direct
{new_director: row.new_director}]->(m);

LOAD CSV WITH HEADERS FROM 'file:///actor_movie.csv'
AS row MERGE (a:actor {name: row.actor}) MERGE
(m:movie {name: row.movie}) MERGE (a)-[r:act_in
{new_actor: row.new_actor}]->(m);

CREATE CONSTRAINT ON (a:actor) ASSERT a.name IS
UNIQUE;
CREATE CONSTRAINT ON (d:director) ASSERT d.name IS
UNIQUE;
```

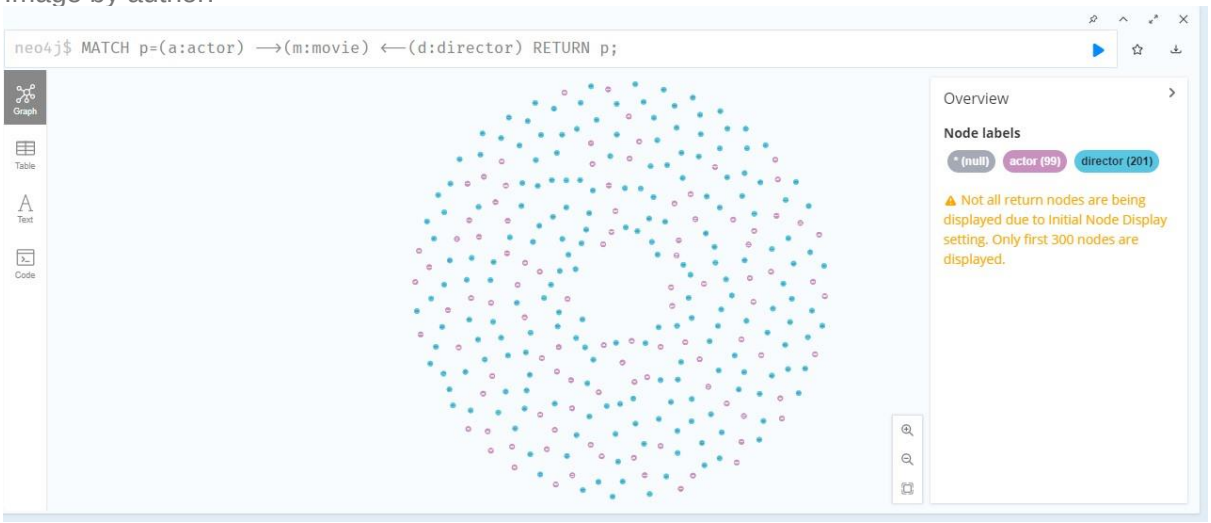
we can check how the three types of nodes (actor, director and movie) are connected:

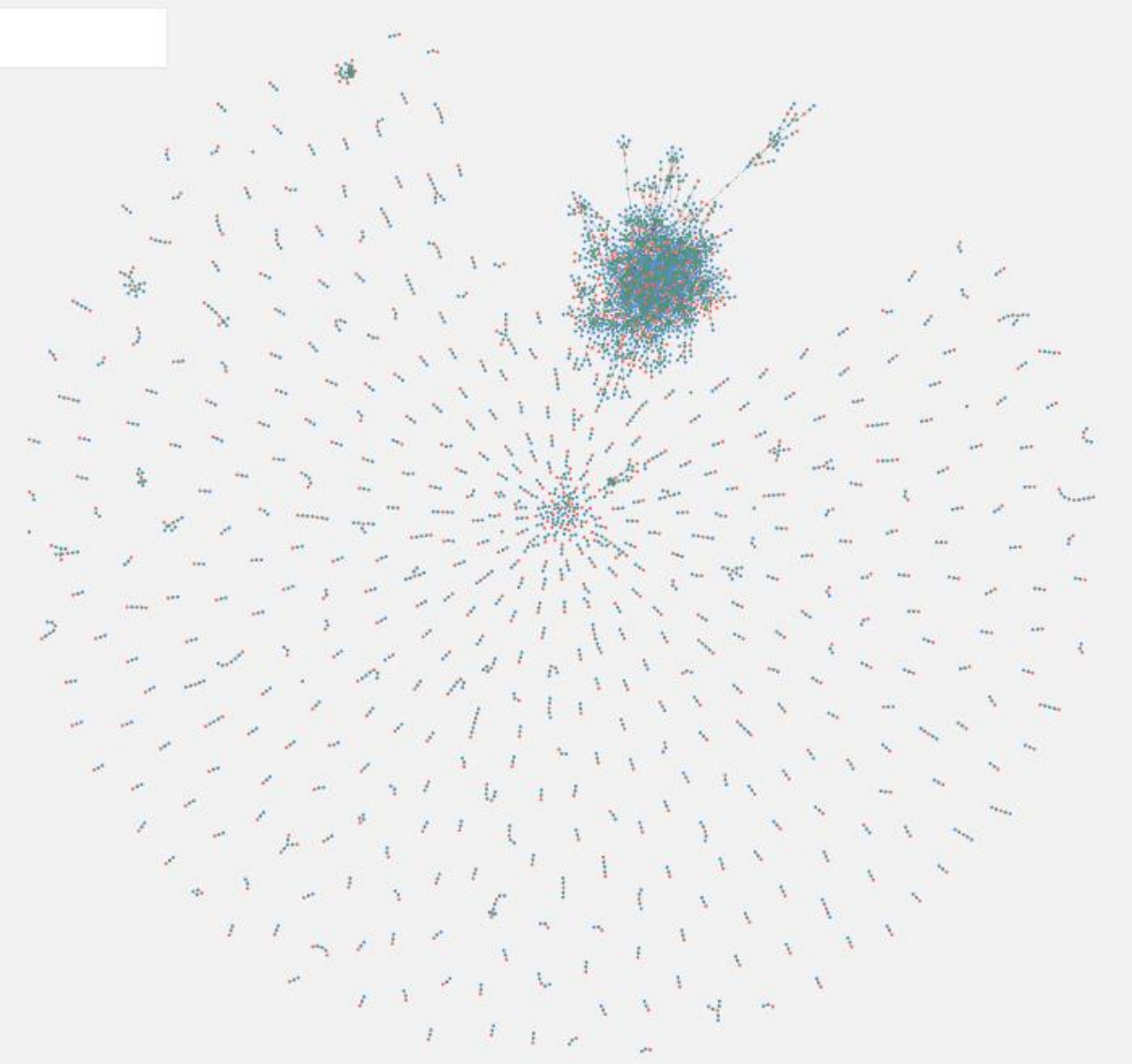
```
call db.schema.visualization()
```



2. Overview

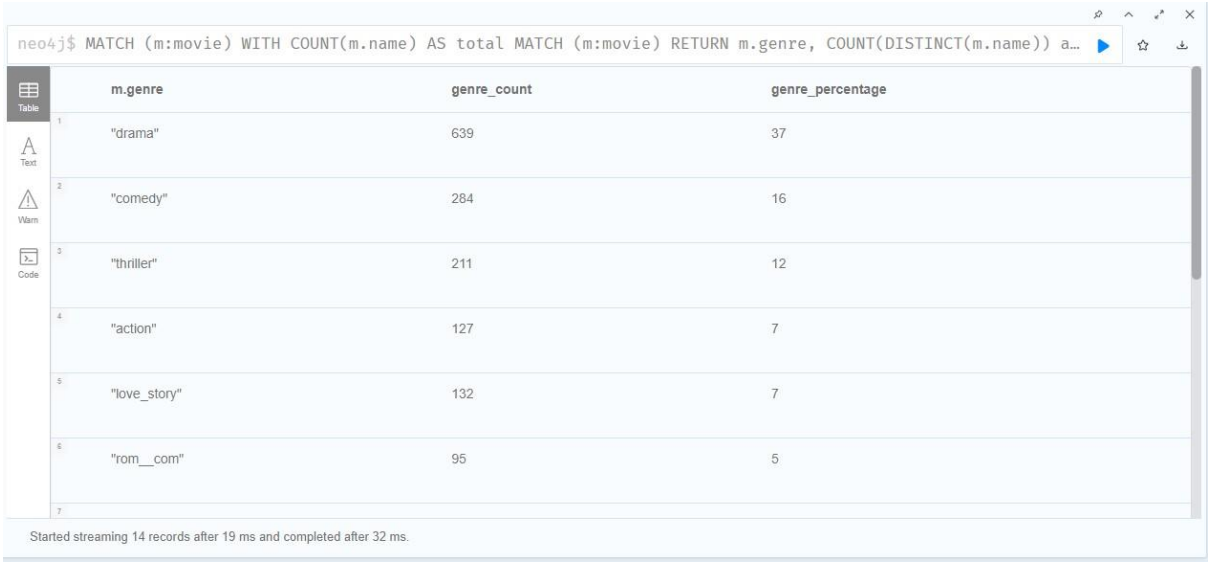
The topological overview of the Bollywood dataset. Green: movie; Red: actor; Blue: director. Image by author.





It is clear that this large cluster is held together by prolific Bollywood actors such as the Three Khans, Akshay Kumar, Emraan Hashmi and directors such as Vikram Bhatt and Mohit Suri.

We can also have a look at the genre distribution in the dataset:



The documentation from Neo4j explains that [WITH is like the pipe operator] in Bash. In the query above, I calculated the total number of films with the WITH statement and then used it in the percentage calculation. The results suggested that dramas, comedies, and thrillers are the top three movie genres in Bollywood.

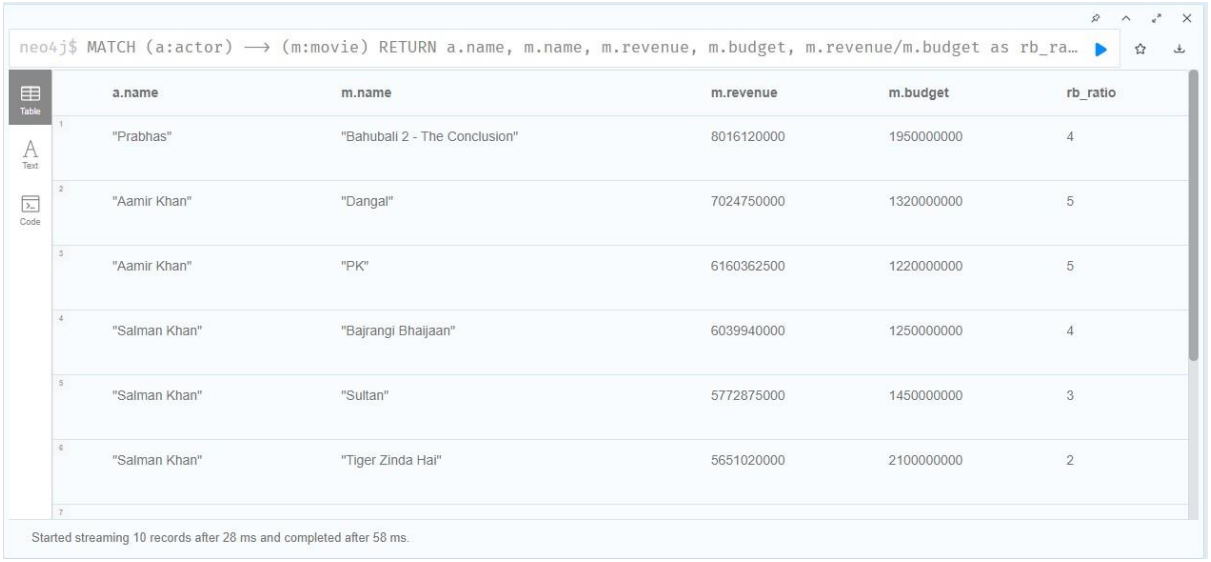
3. Get some statistics

```
MATCH (a:actor {name: "Aamir Khan"}) --> (m:movie)
RETURN sum(m.revenue);

//returns 29030895000
```

we can see the top earners:

```
MATCH (a:actor) --> (m:movie)
RETURN a.name, m.name, m.revenue, m.budget,
m.revenue/m.budget as rb_ratio
ORDER BY m.revenue DESC LIMIT 10;
```



neo4j$ MATCH (a:actor) ⟶ (m:movie) RETURN a.name, m.name, m.revenue, m.budget, m.revenue/m.budget as rb_ra…

Table | Text | Code

	a.name	m.name	m.revenue	m.budget	rb_ratio
1	"Prabhas"	"Bahubali 2 - The Conclusion"	8016120000	1950000000	4
2	"Aamir Khan"	"Dangal"	7024750000	1320000000	5
3	"Aamir Khan"	"PK"	6160362500	1220000000	5
4	"Salman Khan"	"Bajrangi Bhaijaan"	6039940000	1250000000	4
5	"Salman Khan"	"Sultan"	5772875000	1450000000	3
6	"Salman Khan"	"Tiger Zinda Hai"	5651020000	2100000000	2
7					

Started streaming 10 records after 28 ms and completed after 58 ms.

We can see the list of the biggest box-office bombs:

```
MATCH (m:movie)
RETURN m.name, m.revenue, m.budget, m.revenue -
m.budget AS profit
ORDER BY profit LIMIT 10;
```

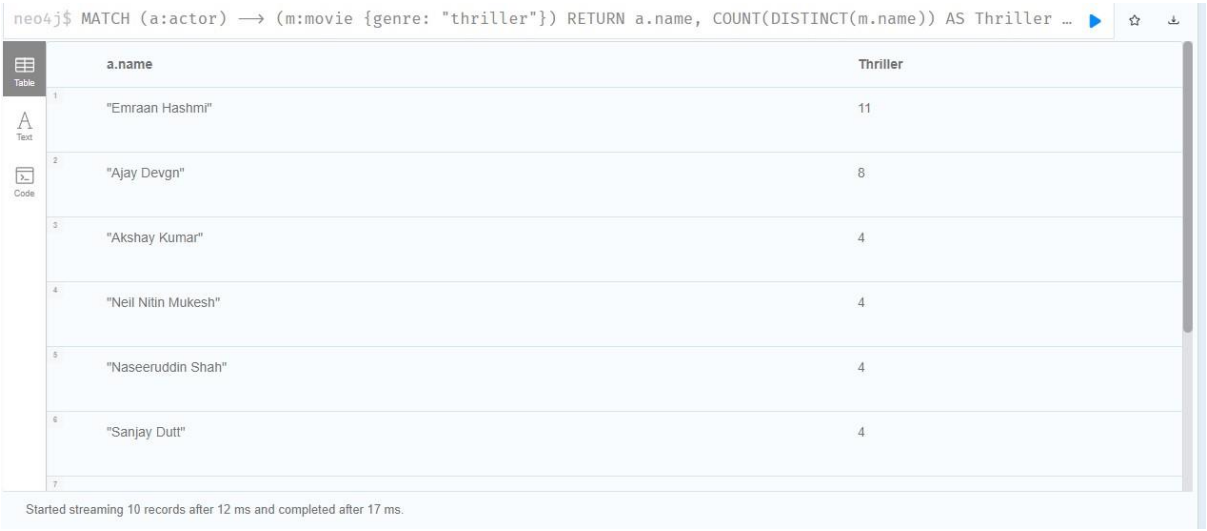
neo4j$ MATCH (m:movie) RETURN m.name, m.revenue, m.budget, m.revenue - m.budget AS profit ORDER BY profit LI…

Table | Text | Code

	m.name	m.revenue	m.budget	profit
1	"Bombay Velvet"	431365000	1180000000	-748635000
2	"Broken Horses"	10130000	600000000	-589870000
3	"Mirzya"	134665000	630000000	-495335000
4	"Jagga Jasoos"	868572500	1310000000	-441427500
5	"Zanjeer"	221475000	600000000	-378525000
6	"Mohenjo Daro"	1040750000	1380000000	-339250000
7				

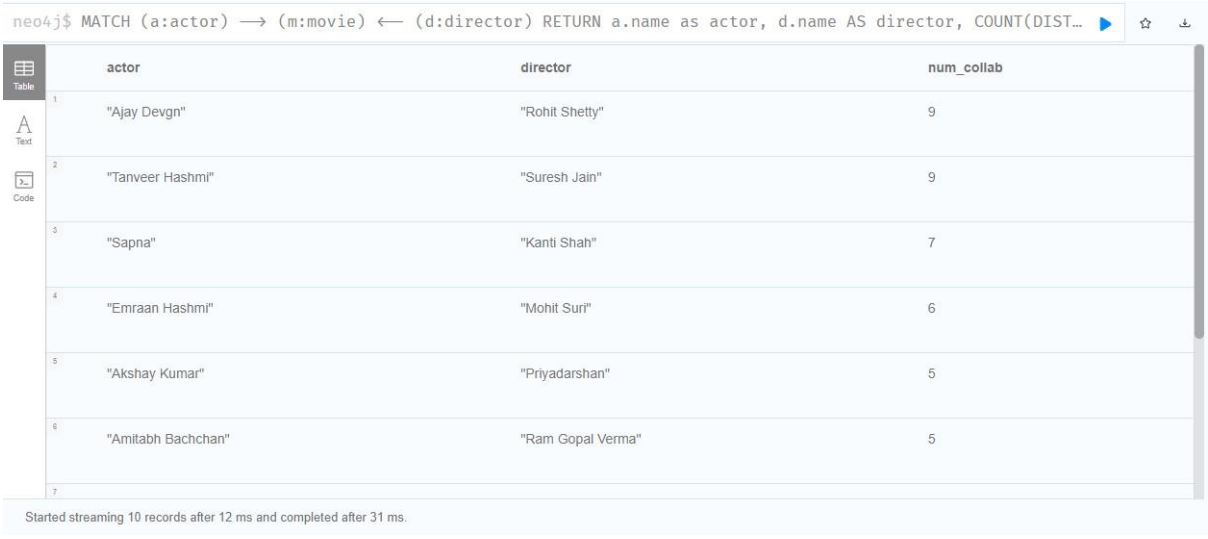
Started streaming 10 records after 11 ms and completed after 36 ms.

Afterwards, we can also see which actors have the most thrillers or horror movies under their belt.



Finally, let's see how often some actor-director duo worked together.

```
MATCH (a:actor) --> (m:movie) <-- (d:director)
RETURN a.name as actor, d.name AS director,
COUNT(DISTINCT(m.name)) AS num_collab
ORDER BY num_collab DESC LIMIT 10;
```



4. Community detection

the topological overview by Neo4j Bloom has shown us a large cluster centered around some of the biggest names in Bollywood. Now the question is whether it is possible to obtain that community through a query? With the help of Neo4j's Graph Data Science Library (GDS), the answer is a resounding "Yes".

To use WCC, you need to enable the GDS plugin in our project

```
CALL gds.graph.create.cypher(
    'bollywood-graph',
    'MATCH (n) RETURN id(n) AS id',
    'MATCH (n)--(m) RETURN id(n) AS source, id(m) AS
target'
);
```

Afterwards, we run WCC. It returns the top 10 largest communities (called "components" in WCC):

```
CALL gds.wcc.stream('bollywood-graph')
YIELD nodeId, componentId
RETURN  componentId, COUNT(componentId) as count
ORDER BY count DESC LIMIT 10;
```

componentId	count
0	1750
209	24
67	24
505	17
206	15
273	15
122	13
310	11
1183	11
68	10

We can see that the Component 0 has as many as 1750 nodes. However, I have found that some nodes were double-counted. Let's see what the nodes are with the DISTINCT function:

```
CALL gds.wcc.stream('bollywood-graph')
YIELD nodeId, componentId
WHERE componentId = 0
RETURN  DISTINCT(gds.util.asNode(nodeId).name) AS
name, componentId ORDER BY name
```

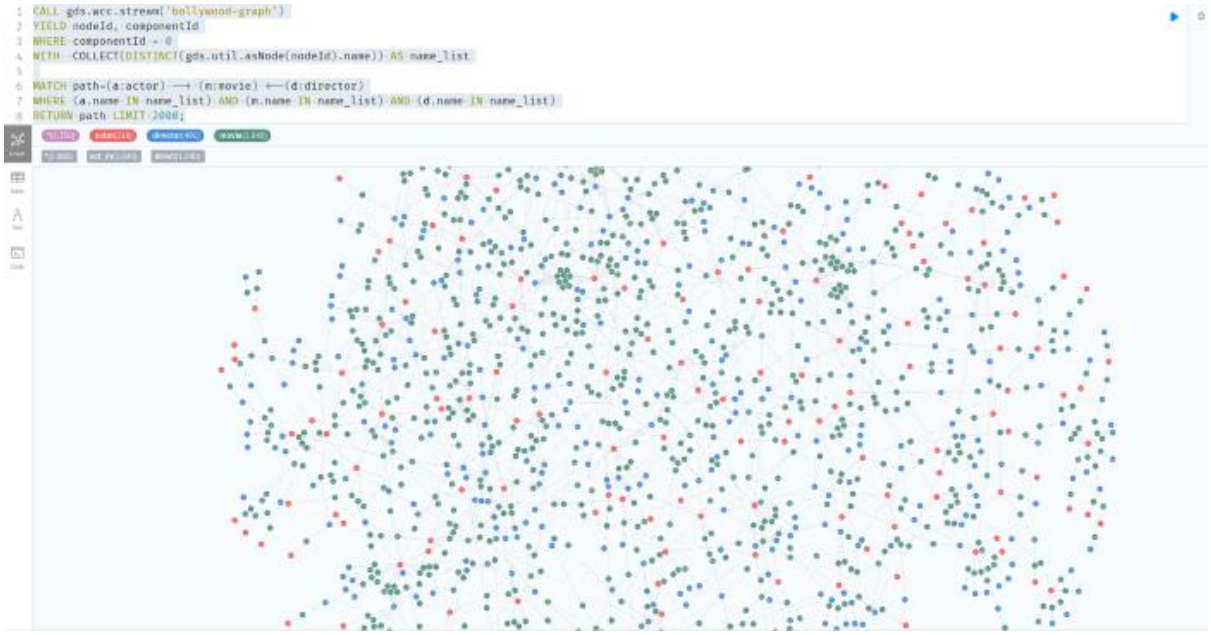
	name	componentId
2	...And Once Again	0
3	10ml Love	0
4	13B	0
5	15 Park Avenue	0
6	1920	0
7	1920 - Evil Returns	0
8	1920 London	0
9	1971	0
10	2 States	0
11	3 Bachelors	0
12	3 Idiots	0
13	3 Nights 4 Days	0
14	36 China Town	0
15	3:00 AM	0
16	3G	0
17	404	0
18	7 1/2 Phere	0
19	7 Khoon Maaf	0
20	8 x 10 Tasveer	0
21	99	0
22	...	...

The query returns 1,734 nodes instead of the original 1,750. We can even show the networks in Neo4j Browser. But first, adjust the visualization parameters in Neo4j Browser

```
CALL gds.wcc.stream('bollywood-graph')
YIELD nodeId, componentId
WHERE componentId = 0
WITH  COLLECT(DISTINCT(gds.util.asNode(nodeId).name))
AS name_list

MATCH path=(a:actor) --> (m:movie) <--(d:director)
WHERE (a.name IN name_list) AND (m.name IN name_list)
AND (d.name IN name_list)
RETURN path LIMIT 2000;
```

The COLLECT function transforms the names into a list. We then do a normal MATCH query and filter the results with that list.



CONCLUSION:

This project shows that Neo4j not only excels in relation-rich data, but it can also aggregate tabular data as easily as SQL. This tutorial showcases primarily the statistical functions in Neo4j. Readers can easily compare the SQL queries with my Cypher queries above and feel the differences in the syntax and the expressiveness of the two languages. According to Wikipedia, the syntax of Cypher is based on ASCII-art. So the queries look very visual and are easy to understand. Combined with its aggregating and Graph Data Science functions, Neo4j can do just anything that a relational database can do and then some.